AD-A269 459

(2)

THE DEVELOPMENT OF A COURSE SEQUENCE IN REAL-TIME SYSTEMS DESIGN

FINAL REPORT

H.H. AMMAR,

AUGUST, 1993

A TECHNICAL REPORT SUBMITTED TO

THE ADVANCED RESEARCH PROJECT AGENCY

Department of Electrical & Computer Engineering

WEST VIRGINIA UNIVERSITY,
MORGANTOWN, WEST VIRGINIA

THE DEVELOPMENT OF A COURSE SEQUENCE IN REAL-TIME SYSTEMS DESIGN

FINAL REPORT

H.H. AMMAR,

AUGUST, 1993

A TECHNICAL REPORT SUBMITTED TO

THE ADVANCED RESEARCH PROJECT AGENCY

contract No. MDA972-92-J-1022.

Department of Electrical & Computer Engineering

WEST VIRGINIA UNIVERSITY,
MORGANTOWN, WEST VIRGINIA

THE DEVELOPMENT OF A COURSE SEQUENCE IN REAL-TIME SYSTEMS DESIGN

FINAL REPORT

H. H. AMMAR,

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING,

WEST VIRGINIA UNIVERSITY

SUBMITTED TO

THE ADVANCED RESEARCH PROJECT AGENCY

TABLE OF CONTENTS

PART IV APPENDIX

PROJECT SUMMARY AND CONCLUSIONS

This project deals with the development of a senior level course sequence in software intensive real-time systems. The sequence consists of a course in real-time software development, referred to in this report as the software course, followed by a course in hardware development and hardware-software interfacing, referred to in the sequel as the hardware course.

The objective of the course sequence is to fill a demand in industry for real-time software engineers. It is aimed at preparing Computer Engineering students with a solid background in both software development and hardware design for an engineering industrial career in real-time systems development. The course sequence emphasizes practical standards, techniques, and tools for system development. Few universities include real-time systems development in their undergraduate Computer Engineering or Computer Science curriculum, forcing engineers to learn real-time methodology on-the-job. Real-time applications are increasing in complexity. Real-time designs have several performance as well as functional constraints, they have longer design cycles and larger project teams. They require more intricate real-time design work such as hardware interfacing, and often use multiprocessors in distributed environments. The growing complexity of real-time designs makes it difficult for an engineer to learn about real-time methodology on-the-job.

The main topics introduced in the course sequence are summarized as follows:

Real-Time Systems Design I (the Software Course)

The software course is based on using the Ada programming language for detailed design and implementation. It is assumed that students are already familiar with the basics of Ada programming. Advanced topics such as tasking and synchronization are introduced in the course.

Throughout the course, an Integrated Computer Aided Software Engineering (ICASE) tool called Teamwork, by Cadre's Technologies, is used.

The course first introduces, in Chapter 1, the basic concepts and characteristics of real-time systems. Examples of such systems, together with their important characteristics such as concurrency and real-time behavior, are briefly discussed from the cited references in the literature. Next, the phases of the software life-cycle are briefly introduced, in the beginning of Chapter 2, and a standard for software development based on the DOD-STD-2167 standard is discussed. The standard is presented as a general industrial model for software development and not as a standard for projects developed for the department of defence. The inputs and outputs of each phase are briefly discussed using the cited reference. The emphasis on studying an established industrial standard for software development offers the students a practical flavor of the software development process as related to large projects.

In the remaining of Chapter 2, the course concentrates on the analysis and design phases, with emphasis on both the structured approach and the object-oriented approach to software development. Structured analysis based on Ward and Mellor, which was then extended by Hatley and Pirbhai, is studied using examples developed through Teamwork/SA and RT (for structured analysis of real-time systems). The ICASE tool offers the necessary environment for developing data/control flow diagrams, state transition graphs and process activation tables used for obtaining a rigorous specification of the software. The tool can also be used for checking the consistency and completeness of the specification. Object-oriented analysis, based on the information model which is defined using entity relation diagrams supported by Teamwork/IM, is then contrasted with structured analysis. The object-oriented analysis technique discussed is based on the one developed by Shaler and Mellor and supported by Teamwork/OOA.

The traditional structured design concepts are then introduced with more emphasis on new concepts such as reusability and testability or verification. Students were exposed to software reuse through collaboration with industry such as the ASSET Ada repository managed by IBM and SAIC. Members of the technical staff of ASSET were contacted. The software support offered by them was used in the course. Students were given the opportunity to use the ASSET database. A special student database was provided for students to populate it with their own components or assets. In the following page, a student report is given. This report was written by Marcus Speaker, a student in the software course. This report stresses the fact that the importance of software reuse can only be appreciated when a reuse library such ASSET is made available to students.

The demonstration of the importance of the concept of design for reuse plays a nice role in introducing the benefits of object-oriented design. This design approach is studied through the notation and techniques described by Booch and supported by Teamwork/OOD. A homework problem designed to familiarize students with the basic concept of object oriented design is given based on an example of a traffic management system. The differences between the design method using teamwork/OOD and Booch's design method give the students a broader view of object-oriented design.

Detailed design concepts based on an Ada implementation are addressed in Chapter 3. The chapter covers the use of Ada to address the characteristics of quality attributes such as modularity, reliability, maintainability, and reusability. It deals with the characteristics of real-time software in terms of concurrent programming, communication and synchronization, The ASSET program appears to be a fast growing project. With the need for a distributed support system for software reuse, ASSET seems to be providing such a system. While using the ASSET system, I began to see the need for software reuse. However, industry uses reuse software much more than students at the university level.

While working with ASSET I found the catalog service very well organized. The search for software was both fast and efficient. The layout was easy to follow even for a beginner. With a wide selection of software, many routines were available. It would be beneficial to see more software in the fields of Multimedia, Discrete Signal Processing, and Artificial Neural Systems. With a good starting selection of reuse, many of the titles seemed to be for government use only. The ASSET catalog should be more open to the public, even at this early date.

When adding entries to the WVU data base, I felt that the system was intended to be used by someone who knew the system very well. Not having any experience with the system, I encountered many obstacles. To better the present system, several steps should be taken. First, the system needs better documentation. The present documentation consists of one and one half pages on how to insert an entry to the system. Give more explanations, and perhaps a tutorial. Next, I found it time consuming and laborious to go through each of the steps of the insert procedure. Implement a menu driven system, much like the catalog to ease the addition of elements to the database.

In all, the ASSET program is producing acceptable results. I enjoyed using the system, and found the work to be a great learning experience.

resource control, and scheduling. Next it describes detailed design and code generation using Ada Structured Graphs which is supported by Teamwork/Ada. The material assumes that students are already familiar with the Ada programming language. Computer Engineering students at WVU taking this course should have taken, during their sophomore year, an introductory course in Ada (CS 15) as well as a data structures course using Ada (CS 16). Ada tasking, however, is not covered in these courses.

The temporal requirements, which constitute the distinguishing characteristics of real-time systems, are then analyzed in Chapter 4. The chapter deals with practical quantitative methods used to analyze and predict the timing behavior of a large class of real time systems. These methods can be used to assess design tradeoffs and verify or troubleshoot the system timing behavior. The chapter covers quantitative methods based on the theory of rate monotonic scheduling. This set of methods termed as rate monotonic analysis (RMA) was recently the topic of a handbook of real time systems analysis produced for industry practitioners by the Software Engineering Institute and published by the IEEE computer Society.

Tasks and techniques used for software verification and validation steps at the early stages of the software life-cycle are discussed in Chapter 5. The importance of these steps to be coupled with the development phases is emphasized.

The above course was offered as a senior level elective course in the fall of 1992 (CpE 291A). Twenty three students took the course. They were mostly seniors, juniors, and a few graduate students. Some of the graduate students were not familiar with Ada, and therefore, were allowed to use C for implementation purposes. The students reacted very favorably to the course material. For them, it was a big change from the computer science Ada courses they took before mainly due to the heavy emphasis in this course on practical techniques and examples. They no longer had to struggle with abstract programming assignments that are far from the engineering systems design and from problems they encounter in their engineering courses. The only problem with the course was the equipments. The SUN workstations lab, an outdated and poorly maintained lab in the engineering college consisting of 8 SUN 3 workstations, was the only lab available to the students in this course. The lab was used to run the ICASE tool and develop the homework and projects. Some of the workstations were extremely slow in running the ICASE tool and were down most of the time.

The following are students comments obtained from the course teaching evaluations:

" Excellent material for a course "

" This is a course that is important to know! The material and design processes are good assets. I like the idea of teamwork involved (computer applications) "

" Good idea for a class, very important topic"

The students in this course were extremely motivated and worked very hard all semester. They put a concentrated effort into the projects. Analysis and design reviews for the projects were very successful. In the final demonstration day at the end of the semester, the groups demonstrated their projects, and all the students did not have enough sleep the night before. Indeed some did not sleep at all to complete the final touches on their project. Some of these projects are summarized next.

Case Studies

The case studies presented in this project, in two separate binders, were the work of the students in the class. Four case studies are presented and briefly described in the following paragraphs.

The first and second case studies are on a manual docking system for the space station. The manual docking system is to be used to guide a user in the process of docking an orbital research unit (ORU) on the space station. Collision avoidance between the ORU and other objects must be maintained at all times. Tactile feedback will be provided to the user in some type of a graphical display. Two different groups worked independently on the above, one ended with an implementation in Ada and the other in C. The work of both groups are included in volume I of the case studies, and the other case studies are included in volume II.

The third case study is on a fully automated commuter train control system. Again two groups worked on this case study and both ended with an Ada implementation. This case study is more complex than the previous one. The requirements for this study were obtained from an example given in Excelerator/RTS application guide.

The fourth case study is on the development of an Airline ticket dispensing system. The system is to be installed in airports and major hotels to allow customers to obtain flight information and tickets from remote sites without an airline agent. This project was implemented in C.

A group of students used the material learned in this course in their senior project. The project is to analyze the hardware and the software required for a methane monitoring system to be used with mining machinery. The monitoring system is to deactivate the equipment when the methane level reaches two percent or higher than the normal level. The system provides self diagnostic tests for keeping record of methane concentration levels over a period of time. This project is a 3 credit hours senior design course. The students used Teamwork/SA-RT to develop the analysis of this system. This is a good illustration of how the students applied the material learned in the software course in their senior project.

Real-Time Systems Design II (The Hardware Course)

This course concentrates on hardware development and software-hardware integration. The course is a laboratory-oriented course consisting of lecturing material and lab assignments. The lecture notes consist of two parts, the first part deals in detail with the hardware components of a real-time system and the second part is on development and trouble shooting tools.

The first chapter on hardware components starts out with a section on microcontrollers. Two families of Intel microcontrollers are discussed in detail, and details of several other microcontroller families are given in the appendix. The other sections deal with sensors, signal conditioning devices, actuators, and data acquisition boards. The section on the data acquisition boards is important since the lab projects are based on using one of these boards.

The second chapter deals with hardware development steps and tools used for testing, trouble shooting, and verification. Simulation tools, emulator circuits, and state of the art digital analyzers are discussed in this chapter.

Next, an introductory lab handout and five lab assignments and projects are given. The lab is based on using Pcs equipped with data acquisition boards and the Meridian OpenAda software development environment. The labs are software intensive and are based on the material learned in the software course. They contain hardware development with emphasis on software-hardware-integration.

The first two assignments are weekly assignments done by each student, the remaining assignments are group projects of 2 to 4 weeks duration. The first lab assignment consists of a simple Ada programming project aimed at familiarizing the students with the OpenAda environment and supporting packages. The second assignment deals with interrupts, low-level I/O, and interfacing Ada to hardware drivers based on C or assembly language. The third assignment, which is a two week assignment, is a sensor monitoring system used in data logging, real-time display of data, and error checking. The fourth lab assignment is a 3 week project, in which an automated train system is simulated. Hardware components are used to simulate the operation of the doors, climate control, and train in motion. The software is required to show detailed information about the status of the train and allow for controlling many aspects of the train operations. The final project is four weeks long and deals with a space shuttle simulation. This project involves the development of hardware to simulate the motion of the shuttle, thrust, skin temperature, etc. The software is required to provide detailed information about the shuttle such as its trajectory and altitude.

The hardware course is yet to be taught due to the severe lack of funds for lab equipment such data acquisition boards, logic analyzers, and even Pcs. The lab is scheduled to be taught in the next academic year when the equipment needed are available.

REAL-TIME SYSTEMS DESIGN I

COURSE DESCRIPTION AND OBJECTIVES

This course is the first course in the real-time system design sequence. This is a project based course focused on software development with an emphasis on analysis and design of software for real-time systems. The course starts by defining real-time systems and describing their characteristics and unique attributes. The software life-cycle phases are then described in the context of the DOD-2167 software development standard. The analysis and high-level design phases of development are then covered by describing both the structured and the object-oriented techniques. Detailed design and implementation details using the ADA programming language are then addressed followed by a detailed coverage of verification and validation techniques and tools.

A computer-aided software engineering (CASE) tool is used throughout the course. A term project and a set of simple homework assignments are used to assess the students learning process. The term project is to be done in teams of 2 to 4 students and it starts very early in the semester during the second or third week and lasts till the finals week. The homework assignments are the responsibility of each student.

The desired outcomes of this course are listed as follows:

The students should be able to

1. Specify the characteristics of real-time software and identify unique attributes and problems related to the software development for real-time systems using specific examples.

PART I

REAL-TIME SYSTEMS DESIGN I

THE SOFTWARE COURSE

REAL-TIME SYSTEMS DESIGN I

COURSE DESCRIPTION AND OBJECTIVES

This course is the first course in the real-time system design sequence. This is a project based course focused on software development with an emphasis on analysis and design of software for real-time systems. The course starts by defining real-time systems and describing their characteristics and unique attributes. The software life-cycle phases are then described in the context of the DOD-2167 software development standard. The analysis and high-level design phases of development are then covered by describing both the structured and the object-oriented techniques. Detailed design and implementation details using the ADA programming language are then addressed followed by a detailed coverage of verification and validation techniques and tools.

A computer-aided software engineering (CASE) tool is used throughout the course. A term project and a set of simple homework assignments are used to assess the students learning process. The term project is to be done in teams of 2 to 4 students and it starts very early in the semester during the second or third week and lasts till the finals week. The homework assignments are the responsibility of each student.

The desired outcomes of this course are listed as follows:

The students should be able to

1. Specify the characteristics of real-time software and identify unique attributes and problems related to the software development for real-time systems using specific examples.

2. Specify the key differences between the structured and object-oriented approaches for software analysis and design.

3. Use a CASE tool to

   a) correctly analyze the software requirements of a specific system and develop a specification based on structured analysis or object-oriented analysis.

   b) develop and verify a high level design according to the specification obtained in the previous step.

   c) develop a detailed design using Ada Structured Graphs (ASGs) and provide an implementation based on reusable Ada packages.

4. Identify techniques used in the analysis and prediction of the timing behavior of a system based on detailed design. Specify examples of using such techniques for Assessing design tradeoffs and troubleshooting the system timing behavior.

5. Identify tasks and techniques used for software verification and validation at the early stages of the software life-cycle.

The following five chapters discuss in detail the topics to be covered and suggest homework problem for assessing and enhancing the students learning process.

CHAPTER I. INTRODUCTION TO REAL-TIME SYSTEMS

1.1 Definitions

A real-time system (RTS) is a system in which the time at which the output is produced is significant.

The correctness of a RTS depends not only on the logical results produced, but also on the times at which such results were produced. (the system may enter an incorrect state if a correct result is produced too early or too late with respect to some time bounds (also called deadlines).

Hard RTSs are those systems where it is absolutely imperative that responses occur within the specified deadlines. (Examples are aircraft control, air traffic control, etc.)

Soft RTSs are those systems were response times are important but the system will still function correctly if deadlines are occasionally missed. (Examples are data acquisition systems in which data can be buffered).

1.2 Characteristics

The following characteristics are often found in many RTSs, however, a system does not need have all these characteristics to be a RTS.

-- A RTS is used within a larger system to provide control and computation functions. Such systems are called "embedded computer systems". They often contain devices that act as the senses (e.g., heat sensors, or light sensors), and devices that act as the effect of physical changes (e.g., mechanical, electromechanical, and electronic actuators).

-- RTS systems often require concurrent processing of multiple inputs. This involves Correlated processing of multiple inputs over the same time interval (e.g. an industrial process control system might be required to correlate values of temperature, pressure, and a concentration of a chemical reaction to perform simultaneous adjustments of heaters and valves to maintain a reaction in the desired state).

-- The time scales of many real-time systems are fast by human standards. The complex Devices monitored or controlled often operate in fast time scales (e.g., for an automobile cruise control system to maintain a smooth ride with only small variations from the desired speed, the actual speed must be monitored many times per second).

-- The precision of response required for RTSs is greater than that required by other systems. An early or a late response may constitute erroneous behavior. A premature shutdown of a chemical plant could cause extensive damage to equipment or environmental harm.

-- RTS systems have higher reliability and safety requirements than that required by other systems. The failure of a system involved in automatic fund transfer between banks can lead to millions of dollars being lost, failure in an embedded system could result in the failure of a vital life-support system.

1.3 Examples:

Examples of RTSs are systems used for process control applications in which a process is monitored and controlled by an RTS (e.g., Industrial process control, manufacturing process control, etc.).

Communications, command and control applications are also examples of RTSs (e.g. real-time audio/video communication, airline reservation systems, medical centers for automatic patient care, air traffic control systems, remote bank ccounting, etc.).

Chapter 1 of [Borko 91] has an excellent introduction to real-time systems. Figure 1.1 in that reference shows several applications of RTSs and the response time range for each application.

1.4 HOMEWORK

* Based on the characteristics of real-time systems (such as concurrency, timing behavior,etc.), specify using examples the unique problems associated with developing software for real-time systems.

1.5 REFERENCES

[Borko 91] Borko Furht et al, Ral-Time Unix Systems: Design and Application guide, Kluwer academic, 1991.

[BURNS&WELLINGS 90] A. Burns, and A. Wellings, Real-Time Systems and their programming languages, Addison Wesley, 1990.

CHAPTER II. INTRODUCTION TO REAL-TIME SOFTWARE ENGINEERING

In this Chapter the development phases in the software life-cycle are first introduced. Then the analysis and design phases are further described based on thc support of the Teamwork

CASE tool notations and components. The well established structured approach as well as the evolving object-oriented approach are covered.

2.1 The Software Life-Cycle

The life-cycle approach for software development describes the stages through which the project passes, and defines the end products in each stage and the activities needed to produce them. It creates a framework for all end products and for testing them as they are produced.

Many engineering studies have been conducted to establish the life-cycle for a system. The life-cycle of a real-time system, in particular, is very complex due to the many distinguishing characteristics mentioned in the above section such as concurrency, reliability, safety, and timing requirements. The preliminary activities establish the system concepts or conceptual basis, then a full-scale development phase starts, and finally, the production and deployment phases follow.

One of the results of these studies is a standard adopted by the US Department of Defense, DOD-STD-2167 (see [DORFMAN&THAYER 90], pp 212-254). The full-scale development phase mentioned in this standard will be considered here further.

Given a set of system requirements, the full-scale development phase starts with requirements analysis and system specification stage, followed by a preliminary design stage, a detailed design stage, the implementation stage, and finally the testing and validation stage. Verification activities are conducted throughout the development phase in order to verify the end product of each stage.

In the requirements analysis and system specification stage a detailed and precise description of the system's functional, timing, and data requirements are developed and documented. The activities in this stage are often carried out in parallel with some high-level design, and the requirements and design activities influence each other as they develop. The system specification document is not a design document. It should set out "what" the system should do without specifying "how". During the creation of this document, errors (or inconsistencies) in the requirements definition (or statement) are discovered and modified accordingly.

The design phase determines how the system is to do the specifications. The two most important activities during design are decomposition and refinement.

Decomposition is the process of partitioning the system into smaller modules. Interfaces between these modules must be precisely specified. Each interface specification provides the module's clients with the information needed to use the module without knowledge of its implementation, at the same time it provides the implementer with the information to implement the module without knowledge of its clients. The interface provides a place for recording design decisions.

Refinement involves working at different levels of abstraction; perhaps refining a module at one level to be a collection of modules at a lower level, hence the term architectural design is used to define this phase of the design process. The design document consists of two parts: (i) architectural design; a description of the system as a whole, and (ii) detailed design; a description of each module.

Design verification is then carried out for checking not just that the design is in accordance with the specification, but that every specification statement is reflected in some part of the design (traceability). Design reviews are carried out to look for logic faults, interface faults, lack of exception handling, and most important non-conformance to the specifications

During the implementation stage, a documented code is obtained from the detailed design document. Unit testing is then carried out on each module to verify that the module correctly implements its detailed design.

Integration testing is then carried out when the modules are put together to determine if the system, as a whole, functions correctly.

2.1.1 HOMEWORK

* Describe in detail the DOD-STD-2167 standard for software development. describe the various documents produced during development, and the reviews conducted.

2.1.2 REFERENCES

[DORFMAN&THAYER 90] Standards, guidelines, and examples on system and software requirements engineering, M. Dorfman and R. Thayer, IEEE Comp. Soc. press tutorial, 1990.

[SOMMERVILLE 92] Software Engineering, Fourth Edition, I. Sommerville, Addison Weley, 1992.

2.2 Requirements Analysis and System Specification

As mentioned above, in the requirements analysis and system specification stage a detailed and precise description of the system's functional, timing, and data requirements is developed and documented. Two major approaches will be discussed, namely the structured analysis approach (also called functional or process oriented approach), and the object oriented analysis approach. The two approaches differ in the way they model the system requirements in order to obtain precise specification.

2.2.1 The Structured Analysis Approach

The main objective of system specification is to develop a conceptual (or a logical) model of the system. In this section, the most widely used method for structured analysis of real-time systems is described. This method was developed by Ward and Mellor [Ward&Mellor 85] and enhanced by Hatley and Pirbhai [Hat&B 88]. It is supported by all Computer-Aided Software Engineering (CASE) tools. The method is an extension of the structured system analysis, data flow analysis, and transaction analysis to real-time systems.

The Ward and Mellor technique relies primarily on three graphs:

1. The Transformation Graphs are basically Data Flow and Control Flow Diagrams (DFDs/CFDs). They depict the processing of information and control in the proposed system.
2. State Transition Diagrams depict the sequence of modes (states) a system follows during its operation.
3. Entity-Relationship Diagrams provide an information model depicting relationships among the data items in the system.

A fully developed specification starts with a context diagram, the highest level transformation graph. This graph places the system in a real-world context. It defines the system as one component and models the interfaces to software and hardware that support and interact with the system.

Other Transformation Graphs (TGs) specify the system at a lower level of detail where a component at a higher level is further specified by a TG at a lower level. The various graphs fit together and form a hierarchy with the context diagram at the top. Nodes in these graphs are either data transformation nodes or control transformation nodes.

Processes specifications (P-specs) are used to define data transformation primitive nodes (those that do not have a lower level TG). P-specs can be defined using pre/post conditions, structured high-level languages, or pseudocode.

Control transformation nodes are specified further (C-specs) using State Transition Diagrams (STDs) which show the control flow behind the system control processes. They give the details of the sequence of states at which the activities (processes) defined in the Tgs are to take place. Decision Tables (DTs), Process Activation Tables (PATs), and State/Event Matrices (SEMs) are also used as C-specs.

The information model defined in an Entity-Relation Diagram (ERD) serves as a library of data records and describes the relationship between data elements used in the system. All the information used in the Transformation Graphs, must correspond to data derived in the ERD.

The above graphs represent three views of the system, namely, the process view (described by transformation graphs), the control view (described by the STDs), and the data view (described by the ERDs). When these graphs are complete and fully developed, they provide a complete logical specification of the system.

Several examples are to be described in class to illustrate the above specification technique. Examples can be taken from the set of case studies provided in the case studies volumes associated with this course or from [HATLEY&PIRBHAI 88].

2.2.1.1 HOMEWORK

The following is a simple assignment designed to introduce the student to the CASE tool and structured analysis process described above.

-- Use Teamwork SA and RT to analyze the requirements and obtain the specification of a vending machine which has the following requirements. The system is to do the following:

* Accept objects from the customer in payment for their purchase.
* Check each object, using the object information (such as size, weight, thickness, and edges) to make sure it is not a slug.
* Accept only nickels, dimes, and quarters, and treat any other object as a slug.
* Initiate payment computation or product selection only after a valid coin is detected.
* Accept product selection from the customer, check to see if product is available, if not return coins and notify customer.
* Products variety can change from time to time, hence prices should be changeable.
* Return back payment if customers decides not to make a selection.
* Dispense the product to the customer if all conditions are satisfied, i.e., if product is available and amount is sufficient.
* Return the correct change to the customer.
* Make deposited coin available for change.

2.2.1.2 REFERENCES

[WARD & MILLER 85] Structured development for real-time systems, P. Ward and S. Mellor, Yourdon Press, 1985.

[HATLEY & PIRBHAI 88] Strategies for real-time system specification, D. J. Hatley and I. A. Pirbhai, Dorset House, 1988.

2.2.2 Object-Oriented Analysis (OOA)

The OOA approach gives more attention to data specification than structured approach which gives more emphasis to functional or procedural specification. The OOA approach is centered around three general concepts: objects, classes, and inheritance. The object-oriented approach has evolved from the concepts of computer simulation which is based on simulating the activities (functions) performed on some entities (objects) of a system.

Objects are the basic run-time entities in an object oriented system. Objects take up space in memory and have an associated address like a data matrix structure for example. The arrangements of bits in an object's allocated memory space determine the state of the object at any given moment. Associated with each object is a set of functions that define the meaningful operations on that object. Thus, an object encapsulates both state and behavior.

Classes define sets of objects, for example, the class of matrices. Ideally, a class is an implementation of an Abstract Data Type (ADT). An ADT consists of the following parts:

1- A type name (e.g. Matrix),

2- An optional specification of the domain of values for the type (e.g, integer values)

3- A specification of allowed operations (e.g., add, multiply, inverse, transpose, etc.) on that type.

The implementation details of a class are private to the class. The public interface of such a class is composed of two kinds of class methods. The first kind consists of accessors functions that return meaningful abstractions about the instance's state. The other type of methods involves transformation procedures used to move an instance from one valid state to another. An Ada package can be used to implement ADT's.

Inheritance is a relation between classes that allows for the definition and implementation of one class to be based on that of other exis classes (for example the class "square matrix" can be defined based on the class "matrix"). Inheritance is the most important concept that helps us realize the goal of constructing software from reusable parts, rather than hand coding every system from scratch.

Inheritance not only supports reuse across systems, but it directly facilitates extensibility within a given system. Inheritance minimizes the amount of work needed when adding additional features.

The logical model, which is based on teamwork/OOA (see also [SHLAER & MELLOR 88]), consists of class diagrams, object communication diagrams, state transition diagrams (STDs), and timing diagrams. The class diagram is built using an Entity Relationship Diagram defined in the previous section. The class diagram shows the various classes and the relationship between them. The object communication diagram shows the data flow between the classes of objects in the system. Then for each class, an STD is defined showing the states of the class and the function and operation which can be activated in each state. A Data Flow Diagram is defined for each state in the STD and timing diagram is also defined for the activated operations.

The following is an example of a traffic light problem showing the class diagram, the object communication diagram, and a state transition diagram.

2.2.2.1 HOMEWORK

1. Using a simple example describe the main differences between structured analysis and OOA.
2. Describe the difference between the information provided in an entity relationship diagram, and an object communication diagram.
3. Specify the OOA of the vending machine described in the previous section using Teamwork/OOA.

2.2.2.2 REFERENCE

[SHLAER & MELLOR 88] Object-Oriented Systems Analysis, Shaler and Mellor, Prentice-Hall, 1988.

2.3 Software Design

As mentioned above the design phase determines how the system is to accomplish the specifications. Following the methodology used in the requirements phase in the pervious section, structure design is covered first in this section, then object-oriented design using the teamwork notation is introduced. These topics are introduced as language independent. The section is then concluded by a description of Teamwork's Ada Structured Graphs as a design tool.

2.3.1 Structured Design

Structured design is characterized by the development of Structure Charts

(SCs) which are used for modeling the partitioning of tasks into modules, the hierarchical organization of these modules, and the data interfaces between them.

The basic units of a SC are the module, the call, the shared data area, and the couple. The module is an independently callable unit of code. The call is an activation of a module, the shared data represent data that is shared among several modules, and the couple represents an item of data passed between modules. The modules in a SC are combined in a tree structure representing a true hierarchy where a module at a lower level may be called by more than one parent.

Each module declared in the SC must be accompanied by a module specification (M-specs). The notation used in the M-specs are similar to those used in P-specs such as pre/post conditions, structure languages, and pseudocode. The amount of detail provided in the M-specs should depend on the destination programming language.

It is important to discuss and understand ways in which a SC can be developed from specifications represented by transformation graphs. Since the top of the hierarchy in the SC is responsible for controlling the decisions of the activities in the task, control transformation structures can be translated into upper-level module structure. Data transformations connected to the control transformations will become lower-level modules.

In cases where no control transformation has been allocated to the task, or when the module-structure of data transformations is needed to be further refined, two techniques are proposed. The first technique is called transform-oriented analysis. In this case the data transformation nodes can be divided into three groups. The first group of nodes are concerned with input data processing, the second group perform some operations on the data, and the third group are concerned with output data processing. Three modules can then be defined in the Sc, an input module, a transform module, and an output module. These three modules can then be refined further into sub-modules.

The second technique is called transaction-oriented analysis. In this case the data transformation is transaction driven, that is, depending on the type of the input transaction, an operation is triggered. In this case a module called get transaction , and a module called dispatcher can be defined and then refined further into sub-modules. The dispatcher module checks the transaction type and identifies the required functions needed.

Structured chart refinement can be guided by the need for satisfying design criteria regarding the coupling of pairs of modules, and the cohesion, complexity, and reusability of individual modules.

Coupling is a measure of connection between two modules. This can take the form of call relationship, parameter passing, or shared data area. One of the important design criteria is to reduce coupling between modules. This because a good modular design lends itself to loosely coupled modules. Techniques for reducing coupling include grouping data into data structures, and eliminating control couples.

Cohesion is a measure of internal relatedness of the components of a module. High cohesive modules are sought out in the process of decomposition and refinement. Components of a module that or operate on shared data structures are more cohesive than modules whose components have merely a precedence relationship. The module name should also represent the functions performed by the components of the module. This is termed as external cohesion.

Complexity is another design criteria used in the process of decomposition and refinement. A module should be simple enough to be regarded as a single unit for purposes of verification and modification.

Reusability is now considered as an important design criteria after the emergence of software repositories which provide means of classifying, cataloging, and retrieving software components. Both domain specific and general reusable modules are sought out in the design of current software systems. In developing reusable modules, upper-level modules are likely to be domain specific, whereas lower-level service modules are more easy to generalize. Examples on structured design can be found in recent text books on software engineering.

2.3.1.1 HOMEWORK

1. Use Teamwork/SD to develop a design of the vending machine problem. Give detailed specifications of the all the modules in you design.
2. The use of the concepts of coupling and cohesion to provide guidelines for creating good designs are shown in [NIELSEN & SHUMATE] (pp 137-149) in terms of Ada packages and tasks. Use such concepts in developing an alternate design of the robot controller example shown in Chapter 22 (pp210) of the same reference.

2.3.1.2 REFERENCES

[NIELSEN & SHUMATE 88] Designing Large Real-Time Systems With Ada, K. Nielsen and K. Shumate, McGraw-Hill, 1988.

[SOMMERVILLE 92] Software Engineering, Fourth Edition, I. Sommerville, Addison

Weley, 1992.

2.3.2 Object-Oriented Design

Object-Oriented Analysis specify the system under development as a set of communication objects. The most important part of Object-Oriented Design is to specify the design of each class in terms of its data structures and member functions. To show an Object-Oriented design, four types of diagrams must be created as follows:

1. A diagram for each class to show class characteristics (e.g., data structures and member functions) and interface characteristics,
2. A class structure chart which shows the implementation of class functions. It shows the internal structure of a single class by showing the structure of the modules and the data and control flow within the class,
3. A dependency diagram that shows the usage of dependencies between classes and nonclass functions or subprograms.
4. An inheritance diagram that shows the relationship between base classes and their derived classes.

The notation used in Teamwork/OOD was developed on top of Ada Structured Graphs to be described in the next chapter. This notation has been tailored to facilitate the generation of C++ code from the design diagrams.

Booch developed a notation in [BOOCH 90] which can be used with any object-oriented programming language. The notation can also be used for both analysis and design. In fact the notation has some resemblance of the one described above in OOA. It consists several diagrams as described as follows: a class diagram which defines the classes in the system and their relationships (similar to the entity relationship diagram of OOA); a State Transition diagram for each class in the system describing the class dynamics of the system by associating each transition with an action; an object diagram showing the objects in the system and their relationship (similar to the object communication diagram of OOA); a timing diagram specifying the time-ordered events in the system; a module diagram used to show the allocation of classes and objects to modules; and a process diagram describing the allocation of concurrently active processes to processors. Several case studies describing the use of the design method and notation, and the implementation in several languages, are provided in the reference (see [BOOCH 90], pp 222-470).

2.3.2.1 HOMEWORK

* Read chapter 12 in the reference (see [BOOCH 90], pp 444-470) which describes the analysis and design of a traffic management system and its implementation in Ada. Describe the differences between the analysis produced using teamwork/OOA, and the analysis used in this chapter. Comment on the difference between the design method using teamwork/OOD and the Booch design method.

2.3.2.2 REFERENCES

[BOOCH 90] Object-Oriented Design with Applications, Grady Booch, Benjamin/Cummings, 1990.

CHAPTER III. ADA AND REAL-TIME SOFTWARE ENGINEERING

In this chapter the use of Ada to engineer better software is addressed. The first section covers characteristics of quality attributes such as modularity, reliability, maintenance, and reusability. The second section deals with the characteristics of real-time software in terms of concurrent programming, communication and synchronization, resource control, and scheduling. The third section describes detailed design and code generation using Ada Structures Graphs which are supported by Teamwork/Ada. This Chapter assumes that students are already familiar with the Ada programming language. Students at WVU taking this course should have taken, during their sophomore year, CS 15 which is an introductory course in Ada as well as CS 16 which a data structures course using Ada. Ada tasking, however, is not covered in detail in these courses.

3.1 Ada and Software Engineering [Schach 90]

Ada was designed to support the principles of modern software engineering. *Modularity:* Ada supports structured programming, the program units can be separately compiled with no sacrifice in compiler checking. Programs compiled as separate compilation units, eliminating module interface errors. Both top-down and bottom-up compilations are supported. Top-down compilation allows the high-level structure of the system to be compiled first, using stubs to substitute for implementations of low-level program units. Bottom-up compilation allows the low-level units to be compiled first providing an easier approach for implementing higher-level modules. Ada goes beyond separate compilation of modules; its compilation units include procedures, functions, packages, tasks, and generics.

Reliability: Ada supports strong type checking as well as range checking. Faults are detected by the compiler if the fault is in the syntax or static semantics of the code, or otherwise by the Ada run-time routines. Most importantly Ada code is more reliable due to the exception handling capability of Ada. Exception handling provides a simple mechanism for fault tolerance. Ada provides three types of predefined exceptions which are raised automatically by the run-time system. The first is a constraint error exception which is raised when an attempt is made to assign to an object a value outside the range specified by the subtype. The second is a storage error exception which is raised when a program runs out of memory, and the third is a numeric error, raised when an arithmetic operation results in an overflow condition.

Object-Oriented implementation: Ada supports data abstraction, information hiding, and a limited form of inheritance through private types, subtypes, and derived types. Private types are important element in Ada's support for *abstract data types*, and are closely related to packages and generics. By hiding the details of the actual data structures that are used to implement it, a user of an abstract data type can only use the operations supplied by the component designer. The *subtyping* mechanism is needed when objects or parameters need to be of the same base type, but are designed to have different subsets of the base type's values. Using *derived data types*, when a type is declared to be derived from another, it inherits an identical set of values as well as all the subprograms declared in the same packet specification as the parent, but is still regarded as a distinct type.

Reusability: Ada was designed to promote reusability. Reusability features are summarized as follows: compilation units can be employed in different contexts when types and subprograms can be passed as parameters; generics (one of the key features which promotes reusability) are essentially parameterized templates; the support of unconstrained arrays as formal parameters and dynamic arrays that allow modules to be used in subroutine libraries. The most promising vehicles for reuse is object-oriented programs. The expected explosion of interest in the practice of reuse and the emergence of the so called software component factories have not yet materialized. However, several Ada repositories such as ASSET, AdaNet, and CARDS are now being developed.

3.1.1 HOMEWORK

This is an optional assignment. Students attempting this assignment will get bonus credit. The assignment involves learning the ASSET reuse repository cataloging procedure, then applying the procedure to catalog an Ada component, and finally writing a brief one page report on your activity.

3.1.2 REFERENCES

[SCHACH 90] Software Engineering, S.R. Schach, Asken Associates, Inc., Publishers, 1990.

3.2 Ada and Real-Time Software [Somerville&Morrison 87]

The goal of this section is to look at the problems of real-time programming systems and how the various facilities in Ada may be used. topics to be discussed are resource scheduling in general, and how task priorities, conditional, delayed and timed-out rendezvous in Ada may be used. Interrupt-driven programming is a necessary part of any real-time environment, as are exceptions and exception-driven programming

Embedded systems require specialized facilities Such as:

1. The need to communicate with external devices,

2. The need to recover from failures and exceptional conditions, and

3. The need to meet some real-time performance criteria.

The need to communicate with devices external to the system in real-time is handled by the interrupt facilities of Ada. The need to recover component failures and other conditions is handled by the exception handling facilities of Ada. Before these concepts are discussed, we should first discuss some basic concepts such as concurrency, scheduling of tasks, entry lists for rendezvous and concept of time in the Ada environment.

3.2.1 Tasks and Task Priorities

Tasks is the compilation unit in Ada which is used to define concurrent processes. An Ada program may consist of many tasks and it is simplest to think of each task as executing on its own individual processor. This ideal situation is often not realizable, since tasks may be created and terminated dynamically whereas the number of processors available to a given system is probably static at any one time.

Task scheduling or the allocation of tasks to processors is not within the control of the user or performed by the compilation system but rather is performed by the run-time environment or system.

Given that there are not enough processors to execute all the tasks, the run-time system scheduler has to decide which task will be allocated a processor. In many timesharing systems, where there is often only one processor, the scheduling strategy used is called time slicing. Each task is allocated to a processor for a certain amount of time during which it may execute. It is then halted and the next task is given the processor for the allocated amount of time which is called a time slice. If ten tasks compete equally for the one processor then each task appears to execute on a processor running at one-tenth of the speed of the real processor. In reality, the speed is less than this since switching between processes takes some time.

A task in Ada may be in one of five states:

-- active: allocated to a processor

-- ready: waiting for a processor

-- blocked: delayed, waiting for I/O or a rendezvous

-- completed: waiting for dependent tasks to terminate

-- terminated: no longer able to execute.

An active process will release a processor when its time slice has been exceeded or when blocked, completed or terminated. When the scheduler decides the task to allocate to a processor, it uses the list of ready tasks. In a round-robin system the list is circular and the next task on the list is given the processor. More sophisticated ordering of the ready list can be accomplished by giving each task a priority.

A task priority in Ada can be assigned using the PRIORITY **pragma**. The pragma is placed in the specification of the task. For examples see the designated reference p267.

3.2.2 Process Communication Using Ada Rendezvous

Process Communication in Ada is accomplished using a rendezvous which is a protocol for synchronizing two tasks. A task may have several entries to which other tasks may have a call. When a task accepts an entry call it executes a sequence of code before ending the accept block at which time a reply is sent to the calling task.

The calling task is suspended until a reply is received, and the called task is suspended at the accept entry until the call is made by the calling task. For examples see reference p243.

Conditional rendezvous, implemented using the select statement, are used to insure that a calling task or a called task should not be delayed to wait for a call or accept. This is important in real-time programming to insure that events are serviced when they exist see reference p269.

Timed rendezvous can be used by a task to check periodically if a call is pending, or an event has occurred. This uses the **delay** statement to suspend the executing task for a certain period of time. See reference p271 for an example.

3.2.3 Resource scheduling

Some control over the relative priority of tasks is achieved using the PRIORITY pragma as mentioned above. However, servicing entry queues is done in FIFO order without considering the priority level. In which case a higher priority task can be blocked waiting for other lower priority tasks in an entry queue to be served.

The above problem can be solved by means of explicit programming using a family of entries facility provided in Ada. See reference section 10.3 for examples. The store controller example in p278, and the elevator simulation example in p279 are two excellent examples which clearly show the above problem and its solution.

3.2.4 Interrupts

It is essential for a real-time system to respond very rapidly to certain stimuli. This is accomplished by means of an interrupt.

A hardware interrupt in Ada is implemented by means of an entry call coming from a hardware task whose priority is guaranteed to be higher than any other task in the program. It may be implemented using an ordinary entry call, a conditional or a timed entry depending on the type of interrupt and on the implementation.

An interrupt must associate a memory address with the entry for the handling routine. When an interrupt at this address occurs the entry is called.

See examples in reference p 281, in Burns and Wellings pp 456-462, and in Nielsen and Shumate pp 189-192.

3.2.5 HOMEWORK

In [BURNS & WELLINGS 91], Chapter 12 (see pages 321-365), the Ada specification of a temperature controller module is given to illustrate the required real-time facilities in a language. Since Ada provides no direct support for deadlines (i.e., there are no predefined exceptions that deal with errors associated with missed deadlines), show how the temperature controller task shown in page 344 can be modified in order to support missed deadline exceptions.

3.2.6 REFERENCES

[SOMMERVILLE & MORRISON 87] Software development with Ada, I. Sommerville and R. Morrison, Addison-Wesley, 1987.

[BURNS & WELLINGS 91] Real-Time Systems and Their Programming Languages, A. Burns and A. Wellings, Addison-Wesley, 1991.

3.3 Ada Structured Graphs (ASGs) [Nielsen&Shumate 88]

Ada Structure Graphs (ASGs) were introduced by Buhr [BUH84] to represent the overall architecture of an Ada program design which will eventually appear as Ada code. It graphically illustrates package structures and the interfaces between tasks that reside in different packages. The ASG diagrams are isomorphic to Ada code specified as a program design language.

The purpose of ASGs is to describe the detailed design of the system. It specifies the overall design architecture in detail including the task structure. It also specifies the inter-package and intra-package control and data flows.

The following conventions are used in an ASG. Large rectangles are used to describe a package structure. Tasks are shown inside packages as parallelograms with entry points as at their edges (entry points are also shown as small parallograms). Entrance procedures are shown as small rectangles at the edges of the package.

The following two pages show an example of using ASGs, developed in Teamwork/Ada, to specify the detailed design of an aircraft monitoring system. The ASG in the first page shows the context diagram of the system which specifies all packages and the dependencies between them. Only the visible tasks and subprograms are shown in the context diagram. Other examples can be obtained from the case studies volume.

3.3.1 HOMEWORK

Refer to the robot controller case study in the designated reference (Case study number 5). Using the structure chart in page 412, develop an ASG for this problem using Teamwork/Ada. Generate code and compare the generated code with the code provided in the reference.

3.3.2 REFERENCES

[NIELSEN&SHUMATE 88] Designing Large Real-Time Systems With Ada, K. Nielsen and K. Shumate, McGraw-Hill, 1988.

CHAPTER IV. ANALYSIS OF TIMING BEHAVIOR

4.1 Introduction

The temporal requirements which constitute the distinguishing characteristic of real-time systems must be analyzed at length. Hard real-time systems have timing constraints that must be satisfied ; soft systems can occasionally fail to perform adequately. Both must be considered within the context of real-time scheduling. This chapter deals with practical quantitative methods used to analyze and predict the timing behavior of a large class of real time systems. These methods can be used to assess design tradeoffs and verify or troubleshoot the system timing behavior.

Formal methods have been extensively used in the research literature in the specification, analysis and verification of the temporal requirements of real-time systems. Although, the state of the art is not quite mature yet to handle a wide-spread use of such methods in large practical projects found in the industrial applications, a maturing set of quantitative methods has evolved based on the theory of rate monotonic scheduling. This set of methods termed as rate monotonic analysis (RMA) was recently the topic of a handbook of real time systems analysis produced for industry practitioners by the Software Engineering Institute [SEI 92].

In this chapter, the RMA techniques outlined in the SEI handbook are briefly summarized and other references containing more information related to an Ada implementation are specified.

4.2 Rate Monotonic Analysis

The timing analysis techniques described in this section are based on scenarios or real-time situations which consist of a collection of event sequence definitions. Each event sequence is characterized by the following: an *event sequence type* which defines the source of the event arrival for example as environmental or internal events; the *event sequence arrival pattern* for example as periodic or irregular events; the current *mode* or state of the system when the event sequence occurred; the *responses* or jobs invoked which specify in terms of sequences of ordered *actions* or tasks (in a sequential, parallel, or selective order) what is done when the event arrives; and finally the timing requirements which define timing restrictions and deadlines on the responses. Actions are further characterized by the resources they use including the CPU, I/O, and data objects, their priority, the resources usage times, and the resources allocation policies.

Let C_i denote the execution time of a response (a job) to a periodic event sequence e_i with period T_i. The utilization U_i associated with the event sequence is defined as $U_i = C_i/T_i$, and the total utilization of a set of event sequences, U, is the sum of all individual event sequence utilization. The total utilization can be used to determine if the event sequences will meet its timing requirements specified in terms the deadlines imposed on the responses to the event sequences. For example, according to the rate monotonic scheduling theory which assigns the highest priority to the action or task with the shortest period, a set of n perfectly pre-emptible, independent periodic tasks will always meet their deadlines if their total utilization is less than or equal to a **utilization bound** defined as $n(2^{1/n} - 1)$. This utilization bound is a theoretically derived number which can also be derived for other more general cases which include dependencies between tasks (e.g., through blocking) and any priority assignment strategy. The utilization bounds provide a way to conduct a fast test on the schedule ability of an event sequence for a given real-time situation.

The response time of a job is affected by other events with higher or lower priority responses which compete for the same resources; events which start with higher priority tasks followed by lower priority tasks,or vice versa; and the execution time of previous jobs in the event sequence which did not complete at the end of their periods. These factors produce preemption effects and blocking effects. For example a lower priority task can be preempted from the CPU by a higher priority task if it is not using any other shared, in which case the higher priority task will be blocked by the lower priority task.

In the following sections, two analysis techniques pertaining to an increasingly complex set of scenarios are described. The first technique can be used to get a fast simplified first-order analysis or tests to check the success of guaranteeing that timing requirements will be met. Unsuccessful tests mean that a more complex technique such as the second technique should be used. The second technique is more involved and it provides a precise schedule ability analysis showing that timing requirements can not be met for a given real-time situation.

4.3 Simple Timing Analysis.

The technique discussed in this section is based on using separate utilization bound for each periodic event when deadlines are within the period. It calculates the effective utilization of an event sequence which includes the effect of other event sequences that can affect the completion time of this event sequence. This effective utilization is then compared with the appropriate utilization bounds and if it is less than the bound, then event sequence is deadline requirements. If the effective utilization is greater than the bound, more precise techniques such as the one described in the next section must be used.

The following steps are applied to each event sequence e_i that has a response time requirement. Let P_i be the priority, and D_i be the deadline, B_i the blocking delay, and u_i the effective utilization of the event sequence.

Step 1. Let H be the set of event sequences which are processed at a priority higher than or equal to that of e_i. Partition the set H into: the set H1 consisting of events with periods greater than or equal to D_i (these events will preempt e_i only once); and the set Hn consisting of events with periods less than D_i. The effective utilization of e_i is calculated as follows:

$$u_i = (\Sigma_{j \in Hn} C_j/T_j) + 1/T_i (C_i + B_i + \Sigma_{k \in H1} C_k)$$

Step 2. The second step determines the utilization bound as follows:

$$U(n,del_i) = (n((2del_i)^{1/n} - 1) + 1 - del_i) \quad \text{, when } del_i > 1/2 \text{, or}$$
$$= del_i \text{ , otherwise}$$

where n is the number of elements in the set Hn plus 1, and del_i is the ratio of D_i/T_i.

4.4 Precise Schedulability Assessment

This technique is based on calculating the worst case completion time for a response in event sequence e_i. The following iterative algorithm is to be applied to each response of interest.

Step 1. Obtain the first estimate a_0 by summing the execution times of the event sequence and all higher priority sequences (it is assumed that event sequences are in priority order, with e1 having the highest priority).

$$a_0 = \Sigma_{j=1 \text{ to } i} C_j$$

Step 2. Obtain the next estimate from the current estimate using

$$a_{n+1} = C_i + \Sigma_{j=1 \text{ to } i-1} |a_n / T_j| C_j$$

Step 3. If (a_{n+1} is less than or equal to D_i) and (a_{n+1} is not equal to a_n) then repeat step 2, else if (a_{n+1} is less than or equal to D_i) then the event sequence is schedulable.

Step 1 in the above algorithm obtains a first estimate of the worst case completion time by neglecting all preemptions. Some of these preemptions are taken into account in step 2. Iterations on step 2 take into account the effect of further preemptions in calculating the worst case completion time. The above technique assumes that deadlines are within the period of events. In this case when the worst case completion time is less than or equal to the deadline schedulability is guaranteed. This technique assumes that deadlines are within the period of the event sequence. More general techniques for arbitrary deadlines and for calculating response time with blocking and step by step examples are described in the designated reference.

4.5 HOMEWOR

1. The following table describes the attributes of three event sequences in a real-time situation:

Event	Arrival period	Execution time	priority	Deadlines
e1	100	40	High	80
e2	150	75	Medium	100
e3	750	65	Low	350

Using the worst case completion time technique of the previous section, determine whether these event sequences will meet their deadlines.

2. In the above table, let 20 and 25 be the blocking delays for events e1 and e2, respectively, using the analysis technique of section 4.2, determine if any of the above events is schedulable.

4.6 REFERENCES

[SEI 93] The handbook of real-time system analysis, The Software Engineering Institute, IEEE Computer Society Press, 1993.

CHAPTER V. VERIFICATION AND VALIDATION TECHNIQUES

5.1 INTRODUCTION

Verification and Validation (V&V) techniques at the early stages of development are used to identify and resolve software problems and high-risk issues early in the software cycle. This results in cost reductions of up to 100:1 in large projects [3] and 10:1 on the smaller projects. Besides major cost savings, there are also significant payoffs in improved reliability, maintainability and human engineering of the resulting software product.

Verification is the process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase.

Validation is the process of evaluating software at the end of the software development process to ensure compliance with software requirements.

In [3] the definition of validation is extended to include a missing activity at the beginning of the software definition process: determining the fitness or worth of a software product for its operational mission.

The scope of this Chapter is to investigate the software tools and technology for performance and reliability evaluation and criticality assessment of the software under development. Analysis and design are the essential first phases in most software development projects [1]. Thus making it very important to consider verification and validation techniques for the early phases.

5.2 VERIFICATION AND VALIDATION CRITERIA

The basic V&V criteria defined in [3] for the requirement/design specifications is completeness, consistency, feasibility and testability. Each of these is discussed below:

5.2.1 Completeness

A specification is said to be complete if it shows the following characteristics [3].

* No ambiguous places, conditions, sentences in the specifications.

* Well defined references to input, outputs and the functions concerned.

* The necessary specification items are not missing.

* The functions, that should be part of the software product, are called for in the specification.

* The products that should be part of the delivered software, are called for in the specification.

5.2.2 Consistency.

Specifications should be consistent from several aspects. Some of these are listed below:

* Internal consistency. Specification should not have any set of components which conflict with each other.

* External consistency. Specification should not have any set of items which conflict with external specifications or entities.

5.2.3 Traceability

Traceability is property of system requirements where specification have clear antecedents in earlier specifications or statements of system objectives. Each of the items should indicate the items in earlier specifications from which it is derived to prevent misinterpretations and embellishments.

5.2.4 Feasibility

If the life cycle benefits of the system specified exceed its life cycle costs the specification are said to be feasible. Based on the feasibility analysis one can identify and resolve any high risk issues earlier in the software development life cycle. The feasibility analysis involves validation of the specified system that it will be sufficiently maintainable, reliable, and human engineered to keep a positive life cycle balance sheet.

* **Human engineering**:

It is based on various conditions, some of which are listed as follows:

* The specified system provides a satisfactory way for users to perform their operational functions.
* The system satisfies human needs at various levels.
* The system helps people fulfill their human potential.

Examples of human engineering considerations are given in [8] and in [9].

* **Resource engineering**:

It is based on the following conditions:

* Systems developed should satisfy the specified requirements, at an acceptable cost in resources.
* The specified system should accommodate cost-effectively, the expected growth in operational requirements over its life-cycle.

Examples of resource engineering considerations are given in [10] and [11].

* **Program Engineering**:

The software program should be engineered in such a way that it will be:

* Cost-effective to maintain.
* Cost-effective from a portability standpoint.

* Able to achieve sufficient accuracy, reliability, and availability to cost-effectively satisfy operational needs over its life cycle.

Examples of these program engineering considerations are given in [12].

* **Risk Issues:**

The high risk issues have to be identified and resolved in advance, otherwise it is very likely that problems will arise when the system is realized. In software requirements and design specifications, technical, cost-schedule, environmental, and interaction effects are some of the sources of the risk.

* <u>Technical risk</u>:

    * Overhead in a multiprocessor operating system
    * Computer security protection
    * Speed and accuracy of new algorithms
    * Performance in "artificial intelligence" domains
    * Man-machine performance

* <u>Cost-schedule risks</u>:

    * Availability and reliability of the underlying virtual machine (hardware, operating system, database management system, compiler) upon which the specified software will be built
    * Percentage stability of the underlying virtual machine
    * Availability of key personnel
    * Strain on available main memory and execution time.

* <u>Environmental risk issues</u>:

* Volume and quality of input data

* Availability and performance of interfacing systems

* Sophistication, flexibility, and degree of cooperation of system users.

5.2.5 Testability

A specification is testable to the extent that one can identify an economically feasible technique for determining whether or not the developed software will satisfy the specifications [3]. Specifications must be specific, unambiguous, and quantitative wherever possible.

5.3 V&V TASKS

Based on the needs of analysis and design phases, V&V tasks can be listed in table 5.1. Some of these tasks are explained in detail in the following sections.

5.3.1. Requirements Analysis

Requirement Analysis is a process to figure out what the customer wants a system to do. The system requirements are normally specified using a graphical and/or textual language. In requirements analysis one can check for syntactical errors in the requirements specifications and then figure out the relationships between system inputs, outputs, processes, and data. During requirement analysis, some or all of the criteria such as completeness, consistency, feasibility and testability are considered.

5.3.1.1. Information input

The input information to requirement analysis is generally the textual form supplied by the customer. In some cases, the information is also supplied in graphical form. Regardless of the format, requirements will specify the inputs to the system and outputs produced by the system. The specifications can also describe the functions or processes to obtain outputs based on the inputs. Requirement document may also set some goals for the expected performance and reliability of the system.

Tasks	Phase(s) of Development Life-Cycle	
	Requirements	High Level Design
Algorithm Analysis	X	X
Analytic Modeling	X	X
Assertion Generation	X	X
Assertion Processing		X
Cause-Effect Graphing	X	X
Control Structure Analysis	X	X
Criticality Assessment	X	X
Data Flow Analysis	X	X
Formal Reviews	X	X
Formal Verification	X	X
Interface Checking	X	X
Regression Testing	X	
Requirements Analysis	X	
Requirements Tracing	X	X
Specification-Based Functional Testing		X
Timing and Sizing Analysis		X

Table 5.1List of V&V tasks for Requirements and High level Design phases

5.3.1.2. Information output

Information output from the process of requirement analysis is as follows:

* Error reports showing syntactical errors or inconsistencies in the specifications.
* Representation of the system indicating static relationships among system inputs, outputs, processes, and data.
* Detailed representation of relationships between different data items.
* A mechanism for simulating the requirements using the generated system representation including the performance and timing requirements.

5.3.1.3. Outline of method

During requirement analysis, results are provided to the user who then interprets them. Characteristics like completeness and consistency are analyzed using various techniques (e.g. simulations, cross referencing), feasibility analysis is performed to make sure the system is worth developing. Input data for other tasks e.g. criticality analysis is prepared.

A data base is developed to maintain and manage the original set of requirements and the subsequent changes in the set.

5.3.1.4. Effectiveness

Requirements analysis tools are very effective for maintaining accurate requirements specifications. Requirement analysis is essential for medium to large systems with a large number of requirements. As the tools and techniques for requirements analysis develop further and their prices become more competitive, small system designs will go through the same rigorous process of requirements analysis.

5.3.2. Requirements Tracing

It is the process where requirements can be traced to customer's statements or from the design components. Thus one can verify that the software system is linked to each requirement. It is an important task as it is used to verify that each requirement is addressed properly and that the software testing will produce adequate and appropriate responses to stimuli based on the requirements.

5.3.2.1. Information input

A set of system requirements serves as input to the process of requirement tracing.

5.3.2.2. Information output

The information obtained from the analysis mainly consists of the correspondence found between the requirements of a system, software specification and the software modules (in high or low level design).

5.3.2.3. Outline of method

Two major tasks can be accomplished through requirements tracing. These are given as follows:

* Task #1: This ensures that each specified requirement of a system is addressed by an identifiable element of the system software.

* Task #2: This ensures that the testing of software produces results which confirm that the system responds adequately to each of these requirements.

Test evaluation matrices can be used in requirement tracing. These matrices identify the requirements of a system which have been appropriately addressed and which have not. There are two kinds of test evaluation matrices as given below:

* Matrices which identify the mapping that exists between the requirement specifications of a system and the modules of that system.
* Matrices that show the mapping between the modules of a system and the set of test cases performed on the system. This matrix determines the modules being invoked by each test case.

The two matrices should be used together to get maximum benefit. The first matrix is useful to analyze the functional requirements of a system. Whereas, the second matrix is useful in analyzing the performance, interface, and design requirements of the system, in addition to the functional requirements. Both are often used in support of a more general requirements tracing activity, that of preliminary and critical design reviews. This procedure is used to verify the traceability of requirements to the design of the system.

5.3.2.4. Effectiveness

This is very effective technique in discovering errors during the design and coding phases of software development, and to verify completeness, consistency, and testability of software. This technique also helps in retesting software by clearly indicating which modules must be rewritten and retested, when a system requirement is changed.

5.3.3. Assertion Generation

Assertion generation is the process to capture the functional properties of a program using an assertion language, for insertion into the various levels of program specification. Assertion generation can be the basis to many verification techniques. These other techniques can then utilize the embedded assertions to compare the actual functional properties of the program with the intended properties.

5.3.3.1. Required input

The input required for assertion generation are the specifications of the desired functional properties of the program. In a relatively simple form, this process breaks down to a preconditions and a post conditions. Preconditions is a set of conditions which have to be true before a module or a process can start performing its intended function and post condition has to be fulfilled before the process can terminate.

5.3.3.2. Information output

The assertions are expressed in a notation called the assertion language. This notation includes higher level expressive constructs quite similar to the programming language. The assertion language is equivalent in expressive power to the first order predicate calculus. The assertions which express the functional properties of the program, can then be used as input to a dynamic assertion processor, a formal verification tool, walk through, specification simulators, and inspections, among other V&V techniques.

5.3.3.3. Outline of method

Assertion generation process is concurrent with the hierarchical elaboration of program functions. When a function is identified as being needed, it is usually specified by a set of inputs and a corresponding set of outputs. Input assertion expresses the requirements on the data the function is to use during its processing. Output assertion expresses what is to be true on function termination.

Later, the necessary steps can be identified to implement the function. After each step it can be said that a "part" of the task has been accomplished. That part is necessary for the proper operation of the next step, and so on, until the entire function has been realized. The character of each part can be captured by an assertion in the same way as the description of the entire function. The output assertion for one step represents (at least part of) the input assertion for the following step. Such assertions are called intermediate assertions. Input, output, and intermediate assertions are placed into the specification of the function being implemented at the appropriate points. Thus, the program design language or the source text will include the assertions. V&V tools, such as dynamic assertion processors, can be developed which could use these during their processing. Later at the coding stage dynamic assertion processors can help validate the source assertions during program execution. In this way program behavior can be verified dynamically.

5.3.3.4. Effectiveness

Assertion generation, when used in conjunction with allied techniques like dynamic assertion processing or functional testing, can be an extremely effective V&V technique. Such effectiveness is only possible, however, when the assertions are used to capture the important functional properties of the program.

5.3.4. Analytic Modeling of System Designs.

Analytical modeling is the process of representing software system in a model. Later this model can be used to obtain information about the characteristics of the system such as performance and reliability issues. Both static and dynamic modeling paradigms can be used to capture the system characteristics. The process usually follows top down approach to design through hierarchical levels. It can be applied at early design stages when functional modules are relatively large and where knowledge of their execution behavior may be imprecise. As the design proceeds and the modules are further resolved, the estimates of their behavior and execution resource characterization become more precise. Design/CPN can be used effectively as a tool to evaluate the system models, where system is modeled using colored Petri-nets.

5.3.4.1. Required input

The inputs required are the system specifications, which can be used to develop a functional design of the system. The functional design then can be utilized to evaluate performance and reliability aspects of the system.

5.3.4.2. Information output

Information output of this task may consist of;

* Throughput of the system
* Resource utilization
* Reliability measures of the system
* Critical modules/processes of the system.

Criticality being measured in terms of

* time bounds on the module activation
* effect of individual components on the system performance.

5.3.4.3. Outline of method

Analytical modeling is carried out in various steps as given below:

* Identification of the functional components of the software design to be modeled.
* Identification of the execution characteristics (primarily, execution time estimate) of each functional component.
* An execution flow graph which gives the definition of the order of execution of the various functional components.

* Execution environment specifications which can include information such as operating system overhead and the workload on the system that could potentially impact the particular software under development.

* System execution scenarios which provide the definitions of the external inputs to the model needed for each simulation of the model.

* Performance goals for the total system and components (an example is an upper bound for the mean and variance of the response time for a specified execution environment and scenario).

After the model is complete various tools can be used to simulate the system and view its performance characteristics.

5.3.4.4 Effectiveness

Effectiveness of this technique in predicting performance depends upon the quality of the performance specifications. The specifications' quality generally improves as the design process proceeds further.

5.3.5 Control Structure Analysis

Control structure analysis reveals violation of control flow standards and improper subprogram usage. It can also identify control branches and paths used in test coverage analysis.

5.3.5.1. Information Input

Input to the control structure analysis can take various forms as given below:

* Text of the program or design to be analyzed.

* Control flow specifications.

5.3.5.2. Information output

Output from control flow analysis consists of the following information:

* Error reports

* Program call graph

Violations of the standards are described in the error reports, whereas the structure of the graph in relation to the use of the submodules is shown in the call graphs. In this analysis routines which are never called as well as attempts to call nonexistent routines can be discovered. Structurally "dead" code within each module can be detected and recursion can be identified.

5.3.5.3. Outline of method

The analysis is carried out on the control specification of the system to generate the error reports and the call graphs. The analyst has to go through the reports and graphs, to look at various aspects of the system e.g. process activation and control input utilization.

5.3.5.4. Effectiveness

The technique is reliable for detecting violations/ambiguities present in the control specifications.

5.3.6 Criticality Assessment

In this process, system requirements are classified to determine their relative importance in terms of such factors as performance, mission, safety, complexity, and cost risk. The measure of criticality assessment can be termed as Criticality Factor (CF) [6]. Once the process of assessment is complete, V&V resources can be concentrated where they are most needed. This also helps while developing test plans to slant the testing toward the more critical requirements.

5.3.6.1. Possible Input

The information needed to perform criticality assessment may consist of the following:

* Program specifications

* Detail of functional components of the program

* Hardware components to be interfaced to the functional modules of the system.

Based on the program specifications, the link between software and hardware components can be analyzed to explore their mutual dependencies. Designer/analyst can then look into critical components based on the safety, life support systems etc. to allocate the CF to each of these components/modules. Moreover there are software modules which are invoked more often than others during program execution. Such modules become more critical from reliability and performance aspects.

5.3.6.2. Information output

Output of the criticality assessment process can be in the form of charts, tables or matrices where critical software/hardware modules are listed along with their CF's.

5.3.6.3. Outline of method

The application of requirement criticality assessment is very much dependent upon the ability to recognize, select, and appropriately define and scale the weighing factors. The criticality assessment methodology consists of several related elements as:

* A numerical figure of merit called the criticality factor (CF) for gauging the relative criticality of each software requirement

* The computation of CF for each of the requirements by V&V analysts in terms of a criticality assessment matrix

* A set of precise and unambiguous instructions supplied to each V&V analyst for determining appropriate entries to the criticality matrix for each of the components of CF
* A set of CF component weighing factors established by V&V and approved by the customer
* A categorical ranking of the CFs, performed by consensus, ordered by the potential for high project payoff leverage when compared to the amount of project resources to be expended
* Concentrated verification through the specification analysis phase of software development of each highly ranked software component in a test configuration thread
* Critical concurrency analysis of each subfunction within each functional data flow.

Risk and complexity are the two issues which form the basis of the system criticality. Risks are evaluated by determination of the adequacy of the automation decisions and of the software/hardware interfaces on the requirements level. Secondly the risks are evaluated by judgmental evaluation of the type and degree of potential for described categories of failure contingencies. Failure mode effects analysis (FMEA) approach becomes necessary to provide additional depth to the quantification[6]. Analysis of the effects that failures, including software errors, can have on the system often requires an in-depth study. Thus, FMEA and criticality assessments work together to provide a way to best utilize the available V&V resources. Complexity is related to the cumulative nature of implementing multidisciplinary software requirements. Complex systems contain interfaces with many subsystems at the same time. Therefore it is necessary to measure the degree to which a given software requirement may impact the baseline system performance requirements if problems occur during program executions. This measure reflects the concept that criticality is directly proportional to the number of connectivities to different subsystems, and that certain subsystem disciplines potentially affect the system performance more than do other disciplines . Balancing these system-related criticality issues is a set of functional issues. These issues require analysis of the adequacy of pertinent aspects of single point control, system interlocks, error control, and man/machine interaction. Of special concern is the identification of those modules (hardware, firmware, or software) with real-time implications. In the software regime, real-time critical components are known to consist of system software (the operating system), the applications software, and support software, which spans system and applications software execution.

5.3.6.4. Effectiveness

This technique is very effective in safety, mission critical environments where a high degree of reliability is expected. NASA and its subcontractors do perform criticality assessment analysis on most of the major projects, an example being the space shuttle project.

5.3.7. Formal Verification

Formal verification is a task/process in which formal and rigorous mathematics is used to prove the consistency between an algorithmic solution and complete specification of the intent of the solution.

5.3.7.1 Information input

The inputs required are the system specifications describing the solution and intent of the system implementation. This information is provided in the form of executable design language and assertions language.

Depending upon the rigor and specific mechanisms to be employed in the consistency proof additional information may be required. Also simplification rules and rules of inference may be required as input if the proof process is to be completely rigorous.

5.3.7.2. Information output

The output report may contain one of the three possible outcomes as given below:

* Successfully completed proof of consistency

* Demonstration of inconsistency

* Terminates inconclusively.

In the first two cases, output is the proofs themselves. In the third case, any fragmentary chains of successfully proven reasoning are the only meaningful output.

5.3.7.3. Outline of method

The usual method used in carrying out formal verification is Floyd's Method of Inductive Assertions or a variant thereof. This method entails the partitioning of the solution specification into algorithmically straightline fragments by means of strategically placed assertions. mis partitioning reduces the proof of consistency to the proof of a set of smaller, generally much more manageable lemmas.

Floyd's Method dictates that the intent of the solution specification be captured by two assertions. The first assertion is the input assertion which describes the assumptions about the input. The second-assertion is the output assertion which describes the transformation of the input, which is intended to be the result of the execution, of the specified solution. In addition, intermediate assertions must be fashioned and placed within the body of the solution specification in such a way that every loop in the solution specification contains at least one intermediate assertion. Each such intermediate assertion must express completely the transformations which are intended to occur or are occurring at the point of placement of the assertion.

The purpose of placing the assertions as just described is to assure that every possible program execution is decomposable into a sequence of straightline algorithmic specifications, each of which is bounded on either end by an assertion. If it is known that each terminating assertion is necessarily implied by executing the specified algorithm under the conditions of the initial assertion, then, by induction, it can be shown that the entire execution behaves as specified by the input/output assertions, and hence as intended. For the user to be assured of this, Floyd's Method directs that a set of lemmas be proven. This set consists of one lemma for each pair of assertions which is separated by a straightline algorithmic specification and no other intervening assertion. For such an assertion pair, the lemma states that, under the assumed conditions of the initial assertion, execution of the algorithm specified by the intervening code necessarily implied the conditions of the terminating assertion. Proving all such lemmas establishes what is known as "partial correctness." Partial correctness establishes that whenever the specified solution process terminates, it has behaved as intended. In addition, total correctness is established by proving that the specified solution process must always terminate. This is clearly an undecidable question, being equivalent to the Halting Problem, and hence its resolution is invariably approached through the application of heuristic.

In the above procedure, the pivotal capability is clearly the ability to prove the various specified lemmas. This can be done to varying degrees of rigor, resulting in proofs of corresponding varied degrees of reliability and trustworthiness. For the greatest degree of trustworthiness, solution specification, intent specification, and rules of reasoning must all be specified with complete rigor and precision. The principal difficulty here lies in specifying the solution with complete rigor and precision. This entails specifying the semantics of the specification language, and the functioning of any actual execution environment with complete rigor and precision. Such complete details are often difficult or impossible to adduce. They are, moreover, when available, generally quite voluminous, thereby occasioning the need to prove lemmas which are long and intricate.

5.3.7.4. Effectiveness

This technique is effective in establishing consistency between intent and solution specification. An inconsistency indicates an error in either or both. The amount of detail needs large, complex lemmas. These, especially when proven using complex, detailed rules of inference, produce very large, intricate proofs which are highly prone to error.

Formal verification of actual programs is further complicated by the necessity to express rigorously the execution behavior of the actual computing environment for the program. As a consequence of this, the execution environment is generally modeled incompletely and imperfectly, thereby restricting the validity of the proofs in ways which are difficult to determine.

Despite these difficulties, a correctly proven set of lemmas establishing consistency between a complete specification and a solution specification whose semantics are accurately known and expressed conveys the greatest assurances of correctness obtainable. This ideal of assurance seems best attainable by applying automated theorem provers to design specifications, rather than code.

5.3.8. Interface Checking

During the process of interface checking, consistency and completeness of the information and control flow between components, modules or procedures of a system are analyzed.

5.3.8.1 Information input

Different kinds of system representations with varying level of description can be used as the input to the process of interface checking. Information can be supplied as formal representation of system requirements or formal representation of system design or a program coded in a high-level language.

5.3.8.2. Information output

Output generally contains reports about module interface inconsistencies and errors.

5.3.8.3. Outline of method

Interface checking analysis is performed on system representation in some sort of program design language. PDL (program design language) describes system requirements as a system of inputs, processes and outputs. Both information and control flow are represented. Interface checking consists of ensuring that all data items are used and generated by some process and that all processes use data. Incomplete requirements specification are, therefore, easily detected.

Other tasks can be used to analyze module interfaces based on a design which contains information describing, for each module, the nature of the inputs and outputs. Module calls can be checked against the interface specifications, in the called module for consistency. This produces a consistency report indicating which interface specifications have been violated.

5.3.8.4. Effectiveness

It is very effective procedure to detect errors which can be difficult to isolate if left to testing. Interface checking can be implemented as part of data flow analysis or a requirements/design analysis.

5.3.9. Regression Testing

In regression testing, a set of test cases is used to perform comprehensive testing of the system's functions. Test cases can be generated using various techniques based on static/dynamic modeling or cause-effect graphing. The set of test cases is maintained and made available throughout the life cycle for testing various stages of development.

5.3.9.1. Information input

A set of system test cases is maintained and made available throughout the entire life of the system for testing purposes. The test cases should be complete enough so that all of the system's functional capabilities are thoroughly tested. Samples of the actual/expected output for each test case is supplied and maintained.

5.3.9.2. Information output

The output from regression testing is simply the output produced by the system from the execution of each of the individual test cases. When the output from previous acceptance tests has been kept, additional output from regression testing should be a comparison of the before and after executions.

5.3.9.3. Outline of method

Regression testing is the process of retesting the system in order to detect errors which may have been caused by program changes. A set of test cases is developed using functional testing. In case of a change in the system requirements, the effected components are identified. Then only these components need to be tested. After the tests are executed, the actual output is compared with the expected output for correctness. When errors are detected during the actual operation of the system which were not detected by regression testing, a test case which could have uncovered the error should be constructed and included with the existing test cases.

5.3.9.4. Effectiveness

The technique is as effective as the quality of the data used for performing the regression testing. Effectiveness improves a great deal if tests are based on the functional requirements.

5.4 REFERENCES

[1] A.I. Wasserman, P.A. Pircher, "SIGPlan Notices", January 1987, pp 131-142.

[2] B.W.Boehm, "Verifying and Validating Software Requirements and Design Specifications", IEEE Software, Vol 1, Number 1, January 1984, pp 75-88.

[3] IEEE Standard for Software Verification and Validation Plans, ANSI/IEEE Std 1012-1986.

[4] "Requirements Analysis and Design Tools Report April 1992", by Software Technology Support Center (STSC), Hill Air Force Base, Utah.

[5] Stephen R. Schach, "Software Engineering", Aksen Associates Inc. Publishers, 1990.

[6] Robert O. Lewis, "Independent Verification and Validation, A life cycle Engineering Process for Quality Software", John Wiley and Sons, INC., 1992.

[7] U.S. Department of Air Force Regulation 800 -14, volume I, Management of Computer Resources in systems, September 1975.

[8] B. Shneiderman, Software Psychology: Human Factors in Computer and Information Systems, Winthrop Press, Cambridge, Mass. , 1980.

[9] S.L. Smith and A.F. Aucella, Design Guidelines for the User Interface to Computer-Based Information Systems,SESD-TR-83-122, USAF Electronics Systems Division, Bedford, Mass, 1983.

[10] B. W. Boehm, Software Engineering Economics, Prentice-Hall, Englewood Cliffs. N..1., 1981.

[11] D. Ferrari, Computer .Systems Performance EvaluationS, Prentice Hall, Englewood Cliffs, N.J., 2nd ed., 1983.

[12] M. Lipow, B. B. White, and B. W. Boehm, "Software Quality Assurance: An Acquisition Guidebook," TRW-SS-77-07, Nov. 1977.

PART II

REAL-TIME SYSTEMS DESIGN II

THE HARDWARE COURSE

REAL-TIME SYSTEMS DESIGN II

REAL-TIME SYSTEMS DESIGN II

COURSE DESCRIPTION AND OBJECTIVES:

This is the second course in the real-time system design sequence. The course is a laboratory based course which emphasizes hardware design and system integration and testing, i.e., the integration of the developed software and hardware, and testing of the integrated system.. The following are the desired outcomes from the course. The students should be able to

1. Design and implement the hardware for a real-time system using a processor, needed sensors, actuators, and signal conditioning devices.
2. Identify and practice trouble shooting techniques for the implemented hardware prototype.
3. Integrate a tested version of the software with the developed hardware.
4. Identify and practice testing and trouble shooting techniques for the integrated system.

CHAPTER I. HARDWARE COMPONENTS OF A REAL-TIME SYSTEM

1.1 INTRODUCTION

Microprocessor based controllers (or microcontrollers), sensors, signal conditioning devices, and actuators are among the most important hardware components in a real-time system. The topics discussed here will deal with each of these components in a separate section.

1.2 MICROCONTROLLERS (reference text: [PEATMAN 88], and [INTEL 93])

Microcontrollers typically consist of the following components: a processor unit implementing a particular machine instruction set; a ROM holding the application program machine instructions as well as static or initialization data; a RAM holding dynamic data and state information; parallel and serial I/O port as well as A/D and D/A converters for executing the input/output operations for analog or digital data samples; and interrupt handling and timer circuitry for event handling and real-time control.

A processor unit is characterized by its instruction set and operand addressing modes; the size of operands taking part in an instruction; the size of the bus (bus width) which determines the size in bits of the data transferred to/from the processor unit and the maximum memory size to be addressed; and the system clock speed which determines the maximum controller instruction processing speed in million instructions per second.

The ROM, used as a nonvolatile storage for the application program and static or initialization data, is characterized by its size in bytes, and whether it is erasable and programmable (i.e., EPROMs) which requires a special device for erasing and programming, or Electrically erasable and programmable (i.e., an EEPROM) which erases quickly and does not require the use of a separate device. The RAM or data memory is characterized by its size and access time. With the increasing complexity of application programs and data, large memory capacity are needed in order to buffer and process large amounts of data. Therefore, memories are important resources in microcontroller units.

Parallel ports are needed for input and output of digital data and handshaking (or control) signals. These can be configured as output only, input only, strobed input or output, or strobed bi-directional. Serial input/output ports can also be used with serial digital data transmission based on standards such as the RS-232 standard. The configuration of serial I/O ports require the specification of whether the serial I/O is synchronous or asynchronous, and the specification of the number of stop bits in an asynchronous I/O.

Analog data I/O is handled by means of A/D and D/A converters in the microcontroller unit. Analog to digital conversion can be done using one of the following A/D converters: dual-slope A/D converter; successive approximation A/D converter; or flash A/D converters. Dual-slope A/D converters obtain a digital output by integrating the analog input over an interval of time. They can, therefore, reduce the effect of noise in the input signal and produce an accurate output data sample. The conversion time is longer however than other A/D converters. Successive-approximation A/D converters are very common im many controllers. They use a D/A and control circuitry to carry out fast conversions. Flash A/D converters are the fastest. They relay on parallel comparisons of the input voltage with all possible sample values using fast comparators and combinational circuits. They are, however, the most expensive since they need a vast amount of circuitry relative to the other two types of converters.

The Motorola 68HC11 microcontroller family contains a successive approximation A/D with an analog multiplexer to multiplex several input analog channels. It also has a sample and hold device to sample the input signal and hold its value constant for the duration of conversion by the A/D. Several families of controllers from Intel will be discussed in the next section.

The interrupt structure and real-time clock of a microcontroller are the most important components related to real-time control. The interrupt structure allows outside events to control the flow of program execution in the microcontroller. The real-time clock is used to produce events or interrupts at particular time instants. The particularly necessary features of the interrupt structure are as follows: multiple prioritized interrupt lines (i.e., vector priority, selective interrupts enable and disable capabilities, minimized delay which occurs before servicing any of several interrupt requests made at the same time). The programmable clock or timer can be used to control the outputs or inputs of the microcontroller to occur in certain time intervals. The use of a programmable timer is essential since the delay loop approach suffers from many drawbacks such as consuming the processor time and requiring all interrupts to be disabled. This is because the execution of the interrupt service routines can affect the delay loop timings.

1.2.1 The RUPI-44 Family of Microcontrollers

The RUPI-44 family is designed for applications requiring local intelligence at remote nodes, and communication capability among these distributed nodes. The RUPI-44 integrates onto a single chip Intel's microcontroller, the 8051-core, with an intelligent and high performance Serial communication controller, called the Serial Interface Unit, or SIU. This dual control architecture allows complex control and high speed data communication functions to be realized cost effectively. The RUPI-44 family consists of three pin compatible parts:

1. 8344 -- 8051 Microcontroller with SIU; 2. 8044 -- An 8344 with 4K bytes of on-chip ROM program memory; and 3. 8744 -- An 8344 with 4K bytes of on-chip EPROM program memory.

1.2.1.1 Microcontroller with On-chip Communication Controller

Architecture Overview

The 8044's dual controller architecture enables the RUPI to perform complex control tasks and high speed communication in a distributed network environment. The 8044 microcontroller is the 8051-core, and maintains complete software compatibility with it.

The microcontroller contains a powerful CPU with on-chip peripherals, making it capable of serving sophisticated real-time control applications such as instrumentation, industrial control, and intelligent computer peripherals. The microcontroller features on-chip peripherals such as two 16-bit timer/counters and 5 source interrupt capability with programmable priority levels. The microcontroller's high performance CPU executes most instructions in 1 microsecond, and can perform an 8*8 multiply in 4 microseconds. The CPU features a Boolean processor that can perform operations on 256 directly addressable bits. 192 bytes of on-chip data RAM can be executed to 64K bytes externally. 4K bytes of on-chip program ROM can be extended to 64K bytes externally. The CPU and SIU run concurrently. See Figure 1.

The SIU is designed to perform serial communications with little or no CPU involvement. The SIU supports data rates up to 2.4 Mbps, externally clocked, and 375 Kbps self clocked (i.e., the data clock is recovered by an on-chip digital phase locked loop). SIU hardware supports the HDLC/SDLC protocol: zero bit insertion/deletion, address recognition, cyclic redundancy check, and frame number sequence check are automatically performed.

The SIU's Auto mode greatly reduces communication software overhead. The AUTO mode supports the SDLC Normal Response Mode, by performing secondary station responses in hardware without any CPU involvement.

The Auto mode's interrupt control and frame sequence numbering capability eliminates software overhead normally required in conventional systems. By using the Auto mode, the CPU is free to concentrate on real time control of the application.

The HDLC/SDLC Protocols

HDLC/SDLC Advantages over Async: The High Level Data Link Control (HDLC) is a standard communication link control established by the International Standards Organization (ISO). SDLC (Synchronous Data Link Control) is a subset of HDLC. They are both well recognized standard serial protocols. SDLC is an IBM standard communication protocol. IBM originally developed SDLC to provide efficient, reliable and simple communication between terminals and computers. The major advantages of HDLC/SDLC over Asynchronous communications protocol (Async) are:

1. Simplicity: Data Transparency;
2. Efficiency: Well Defined Message-Level Operation; and
3. Reliability: Frame Check Sequence and Frame Numbering.

The SDLC reduces system complexity. HDLC/SDLC are "data transparent" protocols. Data transparency means that an arbitrary data stream can be sent without concern that some of the data could be mistaken for a protocol controller. Data transparency relieves the communication controller having to detect special characters.

SDLC/HDLC provides more data throughout than Async. SDLC/HDLC runs at Message-level Operation which transmits multiple bytes within the frame, whereas Async is based on character-level operation. Async transmits or receives a character at a time. Since Async requires start and stop bits in every transmission, there is a considerable waste of overhead compared to SDLC/HDLC. Since SDLC/HDLC has well delineated field, the CPU does not have to interpret character by character to determine control field and information field. In the case of Async, CPU must look at each character to interpret what it means. The practical advantage of such feature is straight forward use of DMA for information transfer. SDLC/HDLC also improves Data throughput using implied Acknowledgement of transferred information. A station using SDLC/HDLC may acknowledge previously received information while transmitting different information in the same frame. Up to 7 messages may be outstanding before an acknowledgement is required. Reliable Data transmission is ensured at the bit level by sending a frame check sequence, cyclic redundancy checking within the frame. Reliable frame transmission is ensured by sending a frame number identification with each frame. It means that a receiver can sequentially count received frames and at any time infer what the number of the next frame to be received should be. It provides a means for the receiver to inform the sender of some particular frame to be resent due to errors.

HDLC/SDLC Networks: In both the HDLC and SDLC line protocols, a (Master) primary station controls the overall network (data link) and issues commands to the secondary (Slave) stations. The latter complies with instructions and responds by sending appropriate responses. When a transmission station must end transmission prematurely, it sends an abort character. Upon detecting an abort character, a receiving station ignores the transmission block called a frame. RUPI-44 supported HDLC/SDLC network configurations are point to point (half duplex), multipoint (half duplex), and loop. In the loop configuration, the stations themselves act as repeaters, so that long links can be easily realized. See fig.2.

Frames: An HDLC/SDLC frame consists of five basic fields: Flag, Address, Control, Data, and Error Detection. A frame is bounded by flags -- opening and closing flags. An address field is 8 bits wide in SDLC, extendable to 2 or more bytes in HDLC. The control field is also 8 bits wide, extendable to 2 bytes in HDLC. The SDLC data or information field may be any number of bytes. The HDLC data field may or may not be on an 8 bit boundary. A powerful error detection code called Frame Check Sequence contains the calculated CRC (Cycle Redundancy Code) for all the bits between the flags. There are three types of frames: Information Frame used to transfer data, Supervisory Frame used for control purposes, and Nonsequenced Frame used for initialization and control of the secondary stations.

Zero Bit Insertion (ZBI): It is desirable to transmit data which can be of arbitrary content. Arbitrary data transmission requires that the data field cannot contain characters which are defined to assist the transmission protocol (like opening flag in HDLC/SDLC communications). This property is referred to as "data transparency". In HDLC/SDLC, this code transparency is made possible by Zero Bit Insertion (ZBI), a bit stuffing technique . The flag has a unique bit pattern: 01111110 (7E HEX). To eliminate the possibility of the data field containing a 7E HEX pattern, ZBI is used. It specifies that during transmission, a binary 0 be inserted by the transmitter after any succession of five contiguous binary 1's. This will ensure that no pattern of 01111110 is ever transmitted between flags. On the receiving side, after receiving the flags, the receiver hardware automatically deletes any 0 following five consecutive

1's. The 8044 performs ZBI and deletion automatically.

Non-Return to Zero Inverted (NRZI): NRZI is a method of clock and data encoding that is well suited to the HDLC/SDLC protocol. It allows HDLC/SDLC protocols to be used with low cost asynchronous modems. NRZI coding is done at the transmitter to enable clock recovery from the data at the receiver terminal by using standard digital phase locked loop (DPLL) techniques. NRZI coding specifies that the signal condition does not change for transmitting a 1, while a 0 causes a change of state. NRZI coding ensures that an active data line will have a transition at least every 5-bit times (recall ZBI), while contiguous 0's will cause a change of state.

Thus, ZBI and NRZI encoding makes it possible for the 8044's on-chip DPLL to recover a receive clock (from received data) synchronized to the received data and at the same time ensure data transparency.

1.2.1.2 The RUPI-8044 Architecture

The 8044 is based on the 8051 core. The 8044 replaces the 8051's serial port with an intelligent HDLC/SDLC controller called the Serial Interface Unit (SIU). Thus the differences between the two result from the 8044's increased on-chip RAM (192 bytes) and additional special function registers necessary to control the SIU.

The Memory Organization of RUPI-8044

The 8044 maintains separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long, of which the lowest 4K bytes are in the on-chip ROM. If the EA pin is held high, the 8044 executes out of internal ROM unless the Program Counter exceeds 0FFFH. Fetches from locations 1000H through FFFFH are directed to external

Program Memory. If the EA pin is held low, the 8044 fetches all instructions from external Program Memory.

The Data Memory consists of 192 bytes of on-chip RAM, plus 35 Special Function Registers, the device is also capable of accessing up to 64K bytes of external data memory. The Program Memory uses 16-bit addresses. The external Data Memory can use either 8-bit or 16-bit addresses. The internal Data Memory uses 8-bit addresses which provide a 256-location address space. The lower 192 addresses access the on-chip RAM. The Special Function Registers occupy various locations in the upper 128 bytes of the same address space. The lowest 32 bytes in the internal RAM (locations 00 through 1FH) are divided into 4 banks of registers, each bank consists of 8 bytes. Any bank can be selected to be the "working registers" of the CPU, and can be accessed by a 3-bit address in the same byte as the opcode of an instruction. Thus a large number of instructions are one-byte instructions. The next higher 16 bytes of the internal RAM (locations 20H through 2FH) have individually addressable bits. These are provided for use as software flags or for one bit (Boolean) processing. This bit-addressing capability is an important feature of the 8044. In addition to the 128 individually addressable bits in RAM, twelve of the Special Function Registers also have individually addressable bits.

Operand Addressing: There are five methods of addressing source operands: Register Addressing; Direct Addressing; Register-Indirect Addressing; Immediate Addressing; and Base-Register-Plus Index-Register-Indirect Addressing. The first three of these methods can also be used to address a destination operand. Since operations in the 8044 require 0(NOP only), 1, 2, 3, or 4 operands, these five addressing methods are used in combinations to provide the 8044 with its 21 addressing modes.

Reset: Reset is accomplished by holding the RST pin high for at least two machine cycles (24 oscillator periods) while the oscillator is running. The CPU responds by executing an internal reset. It also configures the ALE and PSEN pins as inputs (They are quasi-bi-directional). The internal reset is executed during the second cycle until RST goes low. It leaves all the internal registers as 00H or 0000H, except SP as 07H, P0-P3 as 0FFH, IP as ***00000, IE as 0**00000, SIUST as 01H, and PCON as *******. The internal RAM is not affected by reset. When VCC is turned on, the RAM content is indeterminate unless VPD was applied prior to VCC being turned off.

The RUPI-44 Serial Interface Unit (SIU)

The serial interface provides a high-performance communication link. The protocol used in for this communication is based on the IBM SDLC. The serial interface also supports a subset of the ISO HDLC protocol. The SDLC/HDLC protocols have been accepted as standard protocols for many high-level teleprocessing systems. The serial interface performs many of the functions required to service the data link without intervention from the 8044's function as a peripheral controller, rather than having to deal with the details of the communication process. Five pins are involved with the serial interface: Pin7: RTS/P16; Pin8: CTS/P17; Pin10: I/O/RXD/P30; Pin11: DATA/TXD/P31; Pin15: SCLK/T1/P35. SIU.

Data Link Configurations: The serial interface is capable of operating in three serial data link configurations: Half-Duplex, point-to-point; Half-Duplex, multipoint (with a half-duplex or full-duplex primary); Loop.

Data Clocking Options: The serial interface can operate in an externally clocked mode or in a self-clocked mode.

In the externally clocked mode, a common Serial Data Clock (SCLK on pin 15) synchronizes the serial bit stream. This clock signal may come from the master CPU or primary station, or from an external phase-locked loop local to the 8044. Incoming data is sampled at the rising edge of SCLK, and outgoing data is shifted out at the falling edge of SCLK.

The self-clocked mode allows data transfer without a common system data clock. Using an on-chip DPLL, the serial interface recovers the data clock from the data stream itself. The DPLL requires a reference clock equal to either 16 times or 32 times the data rate. This reference clock may be externally supplied or internally generated.

Data Rates: The maximum data rate in the externally clocked mode is 2.4M bits per second (bps) in half-duplex configuration, and 1.0M in a loop configuration. In the self clocked mode with an external reference clock, the maximum data rate is 375K bps. For the 8044 operating with a 12 MHz crystal, the available data rates are 244 bps to 62.5K bps, 187.5Kbps, and 375K bps.

Operational Modes: The SIU can operate in either of two response modes: AUTO, or or FLEXIBLE (NON-AUTO) mode.

Frame Format Options: Variations on the basic SDLC frame consist of omitting one or more of the fields. The choice of which fields to omit, as well as the selection of AUTO versus FLEXIBLE mode, is specified by the settings of the following three bits in the Serial Mode Register (SMD) and the Status/Control Register(STS):

1. SMD Bit 0: NFCS (No Frame Check Sequence);
2. SMD Bit 1: NB (Non-Buffered Mode -- No Control Field); and
3. STS Bit 1: AM (AUTO Mode or Addressed Mode).

Standard SDLC Format: The standard SDLC format consists of an opening flag, an 8-bit address field, an 8-bit control field, an n-byte information field, a 16-bit Frame Check Sequence (FCS), and a closing flag. The FCS is based on the CCITT-CRC polynomial ($X^{16} + X^{12} + X^5 + 1$). The address and control fields may not be extended.

HDLC: In addition to its support of SDLC communications, the 8044 also supports some of the capabilities of HDLC. The principal differences between SDLC and HDLC are as follows:

1. HDLC permits any number of bits in the information field, whereas SDLC requires a byte structure (multiple of 8 bits). The 8044 itself operates on byte boundaries, and thus it restricts fields to multiples of 8 bits.
2. HDLC provides functional extensions to SDLC: an unlimited address field is allowed, as well as extended frame number sequencing.
3. HDLC does not support an operation in loop configurations.

SIU Special Function Registers (SIU SFR): The 8044 CPU communicates with and controls the SIU through hardware registers which are accessed using direct addressing. The SIU SFRs are of three types:

1. Three SIU Control and Status Registers: SMD (Serial Mode Register), STS (Status/Command Register) and NSNR (Send/Receive Count Register);
2. Eight Byte Addressable Parameter Registers: the STAD (Station Address Register), TBS (Transmit Buffer Start Address Register), TBL (Transmit Buffer Length Register), TCB (Transmit Control Byte Register, RBS (Receive Buffer Start Address Register), RBL (Receive Buffer Length Register), RFL (Receive

Field Length Register), RCB (Receive Control Byte Register); and

3. The three ICE (In-Circuit Emulator) Support Registers: DMA CNT (DMA Count Register), three byte FIFO, and SIUST (SIU State Counter).

Operation: The SIU is initialized by a reset signal (on pin 9), followed by write operations to the SIU SFRs. Once initialized, the SIU can function in AUTO mode or NON-AUTO mode.

1.2.2 The MCS-96 Family

The Intel MCS-96 family members are all high performance microcontrollers with a 16-bit CPU and at least 230 bytes of on-chip RAM. It easily handles high speed calculations and fast I/O operations. Typical applications include closed-loop control and mid-range digital signal processing. Modems, motor control systems, printers, engine control systems, photocopiers, anti-lock brakes, air conditioner control systems, disk drives and medical instrumentations all use MCS-96 products. All of the MCS-96 components share a common instruction set and architecture. However, the CHMOS components have enhancements to provide higher performance with lower power consumption. To further decrease power usage, idle and power down modes are available on these devices. These microcontrollers contain dedicated I/O subsystems and perform 16-bit arithmetic instructions including multiply and divide operations. The block diagram of the MCS-96 architecture is shown as Figure 1.

The CPU

The major components of the MCS-96 CPU are the Register File and the Register/Arithmetic Logic Unit (RALU). Locations from 00H to 17H are the I/O control registers or Special Function Registers (SFRs). Locations 18H and 19H contain the stack pointer, which can serve as a general purpose RAM when not performing stack operations. The remaining bytes of the register file serve as a general purpose RAM, accessible as bytes, words, or double words.

Calculations performed by the CPU take place in the RALU. The RALU contains a 17-bit ALU, the Program Status Word (PSW), the Program Counter (PC), a loop counter and three temporary registers. The RALU operates directly on the Register File, thus eliminating accumulator bottleneck and providing for direct control of I/O operations through the SFRs.

The Architecture

The MCS-96 supports a complete instruction set which includes bit operations, byte operations, word operations, double-word operations (unsigned 32-bit), long operations (signed 32-bit), flag manipulations as well as jump and call instructions. The Jump Bit Set and Jump Bit *Clear instructions can operate on any of the SFRs or bytes in the lower register file. These fast* bit manipulations allow for rapid I/O functions.

Byte and word operations make-up most of the instruction set. The Assembly Language ASM-96 uses a "B" suffix on a mnemonic for a byte operation, otherwise the mnemonic refers to a word operation. One, two, or three operand forms exist for many of the instructions. Long and double-word operations include shifts, normalize, multiply and divide. The divide instruction functions as a 32-bit by 16-bit divide that generates a 16-bit quotient and 16-bit remainder.

The word multiply operates as a 16-bit by 16-bit multiply with a 32-bit result. Both operations can function in either the signed or unsigned mode. The normalize instruction and sticky bit flag provide hardware support for the software floating point package (FPAL-96).

Addressing Modes

The MCS-96 instruction set supports the following addressing modes:

1. Register-Direct Addressing;
2. Indirect Addressing;
3. Indirect with Auto-increment Addressing;
4. Immediate Addressing;
5. Short-Indexed Addressing; and
6. Long-indexed Addressing.

Each instruction uses at least one of the addressing modes. The register-direct and immediate addressing modes execute faster than the other addressing modes. The register-direct addressing mode provides access to the addresses in the register file and the SFRs. The indexed modes provide for direct access to the remainder of the 64K address space. Immediate addressing uses the data following the opcode as the operand. Both the indirect addressing modes use the value in a word registers the address of the operand. The indirect auto-increment mode increments a word address by one after a byte operation and two after a word operation. This addressing mode provides easy access into look-up tables.

The long-indexed addressing mode provides direct access to any of the locations in the 64K address space. This mode forms the address of the operand by adding a 16-bit 2's complement value to the contents of a word register. Indexing with the zero register allows "direct" addressing to any location. The short-indexed addressing mode forms the address of operand by adding an 8-bit 2's complement to the contents of a word register. The multiple addressing modes of the MCS-96 family make it easy to program in assembly language and provide an excellent interface to high-level languages.

The Peripherals

8X9X Peripherals: The 8X9X peripherals include the following:

1. Standard I/O Ports;
2. Timers;
3. High Speed Input Unit (HSI);
4. High Speed Output Unit (HSO);
5. The Serial Port;
6. Pulse Width Modulator (PWM);
7. A/D Converter; and
8. Interrupts.

8XC196KB Peripherals: The 8XC196KB peripherals include the following:

1. Standard I/O Ports;
2. Timers;
3. High Speed Input Unit (HSI);
4. High Speed Output Unit (HSO);
5. The Serial Port;
6. Pulse Width Modulator (PWM);
7. A/D Converter; and
8. Interrupts.

8XC196KC and 8XC196KD Peripherals: The 8XC196KC and 8XC196KD peripherals include the following:

1. Standard I/O Ports;

2. Timers;

3. High Speed Input Unit (HSI);

4. High Speed Output Unit (HSO);

5. The Serial Port;

6. Pulse Width Modulator (PWM);

7. A/D Converter;

8. Interrupts; and

9. Peripheral Transaction Server (PTS).

8XC196KR and 8XC196KT Peripherals: The 8XC196KR and 8XC196KT peripherals include the following:

1. Standard I/O Ports;

2. Event Processor Array (EPA);

3. Serial I/O Port (SIO);

4. Synchronous Serial I/O Port (SSIO);

5. A/D Converter;

6. Interrupts;

7. Peripheral Transaction Server (PTS); and

8. Slave Port.

8XC196NT Peripherals: The 8XC196NT peripherals include the following:

1. Extended Address Port EPORT);

2. Standard I/O Ports;

3. Event Processor Array (EPA);

4. Serial I/O Port (SIO);

5. Synchronous Serial I/O Port (SSIO);

6. A/D Converter;

7 Interrupts;

8. Peripheral Transaction Server (PTS); and

9. Slave Port.

8XC196MC Peripherals: The 8XC196MC peripherals include the following:

1. On-Chip Peripherals;

2. I/O Ports;

3. Timers and Event Processor Array (EPA);

4. Pulse Width Modulation Unit;

5. A/D Converter;

6. Interrupt Controller and Peripheral Transaction Server(PTS);

7. Waveform Generator.

1.3 SENSORS (reference text: [TOMPKINS&WEBSTER 88])

In this section, typical transducers or sensors which provide inputs and can be interfaced to a microprocessor system are discussed. Several types of sensors for measuring temperature, light conditions, displacement, and flow are briefly described.

A transducer is a device used for determining the value, quantity, or condition for some physical variable or phenomenon which must be monitored. It usually measures the magnitude of some particular phenomenon for control processing such as temperature, pressure, stress, etc.. The measurements consist of energy transfer which is used to produce the required information.

Piezoelectric energy can be converted into electrostatic charge or voltage when certain crystals are mechanically stressed.

Energy can also be converted to a change of resistance of a semiconductive material from the amount of illumination on the semiconductor surface.

Energy can also be converted to a change of capacitance by using a mechanical means to change the capacitive coupling between two elements.

The are several criteria to consider when selecting a transducer. Examples are: the accuracy of the measurements, lower and upper limits of frequency response, power limitations, signal conditioning required, and transducer effect on the measurand.

In the following sections, specific types of sensors are discussed.

1.3.1 Temperature sensors

In this section we discuss IC temperature sensor, thermocouple devices, and resistive temperature sensors.

1.3.1.1 IC Temperature Sensors

IC temperature sensors are based on the temperature-voltage characteristics of semiconductor devices. For example the base-emitter voltage of a transistor varies directly with temperature at a constant collector current.

An IC temperature sensor circuit produces an output voltage proportional to absolute temperature. Examples are the National Semiconductor LX5700, LM135, LM235, and LM335.

The LX5700 has an operating range from -55 to 125 °C and the sensitivity is 10 mV/C. The output voltage is 2.98 at 298 °K, and the accuracy is 3.8 °K which may not be satisfactory for many applications.

The LM135, LM235, and LM335 operate as two terminal zeners and have a breakdown voltage directly proportional to the absolute temperature.

The output of LM335 can be expressed as:

$$v_o(T) = v_o(T_o)\ T/T_o,$$

where T is the unknown temperature and T_o is a reference temperature, both in Kelvin. By calibrating the output to read correctly at one temperature, the output at all temperature becomes correct.

The LM135 has a range of -55 to 150 C, and a maximum error of 1.5 C over a 100 C range when it is calibrated at 25 C. The LM335 has a maximum error of 2 C.

The Analog Devices AD590 is a two terminal IC temperature sensor whose output current is proportional to absolute temperature. It achieves a higher performance in terms of accuracy and linearity than the LM135. The maximum error of the AD590J is 0.3 C over a 100 C range, and that of the AD590M is 0.05 C. The sensor in insensitive to the voltage across it and acts as a high impedance constant-current regulator passing only 1 micro Amp/K for supply voltages between 4 and 30 V.

1.3.1.2 Thermocouple

Thermocouple are two-wire devices consisting of dissimilar metals or alloys joined or welded together at the sensing junction, and terminated at their other end by a reference junction which is maintained at a known temperature called the reference temperature. When a temperature difference exists between the two junctions, a voltage is produced. This is known as the thermoelectric effect which is caused by the contact potentials at the junction. The produced voltage is very small and requires signal conditioning.

The Analog Devices AD2B50 is a thermocouple signal conditioner. It provides circuitry for input protection, isolation, common mode rejection, low-drift amplification, filtering, and cold-junction compensation all in one chip.

The gain, G = 200 KOhm / RG + 1, is determined by a user supplied resistor, of value RG ohms, connecting pins 3 and 5. An integral reference junction sensor is provided for direct thermocouple connection.

1.3.1.3 Resistive Temperature Sensors

Resistive Temperature sensors are based on materials whose electrical resistance change with temperature. These material are either conductive material or semiconductors. Resistive temperature sensors made up of conductive material are called resistance temperature detectors, and those made up of semiconductors are called thermistors.

Resistance temperature detectors (RTDs) use materials such as platinum, nickel, and copper. Currently platinum is the most common since it is less susceptible to contamination. The increase of resistance with temperature for platinum is approximately linear in the range from 0 to 1000 C and follows the following relation,

$$R_T = R_o (1 + aT),$$

where R_T and R_0 are the resistance at temperatures T and 0 C, respectively. And a is a constant.

The resistance of RTD elements range from about 10 Ohm to 25 KOhms. Platinum is used for a temperature range from -267.78 C to 1010 C.

For more information on RTDs and their use in data acquisition cards see [Miller 86].

Thermistors are thermally sensitive resistors made up of semiconductor materials. They usually have high resistivity and a high negative temperature coefficient (i.e., the resistance decreases with increasing temperature). The temperature coefficient can be as large as several percent per degree Celsius which makes the thermistor able to detect changes in temperature that could not be detected with RTDs or thermocouple circuits. This high sensitivity comes, however, with a highly nonlinear resistance-temperature characteristics.

A special type of thermistors have a positive temperature coefficient. These are often called switching thermistors. There resistance-temperature characteristics remains essentially constant as temperature increases until a point at which the curve increases sharply with each slight increase in temperature. These thermistors are frequently used as thermostats to regulate the temperature in a particular environment.

The basic resistance-temperature characteristics of a thermistor is as follows:

$$R \quad R_0 \exp [B(1/T - 1/T_0)],$$

where B is a material constant for the thermistor in degree Kelvin.

For more information on thermistors, their applications, and their interfacing to the IBM PC see [TOMPKINS&WEBSTER 88] pages 207-221.

Special types of temperature sensors are used in applications such as monitoring the high temperature of a nuclear reactor, and in the steel industry to measure the temperature of a moving steel sheet.

An ultrasonic thin-wire sensor is used in nuclear reactors to measure temperatures in the range of 2000 to 3000 C with a maximal error of 30 C. It is based on the temperature dependence of the velocity of sound.

Eddy-Current sensors are used in the steel industry to measure the temperature of a moving steel sheet from 25 to 300 C with an error of less than 3 C. These sensors are based on the eddy currents produced when a high frequency magnetic field cross the surface of a steel plate. The magnitude of the eddy current changes with the steel plate temperature.

Other special temperature sensors include a quartz-crystal sensor, and a nuclear quadruple resonance sensor. For more details see the reference text.

1.3.2 Optical Sensors

Optical sensors are defined as those sensors which are sensitive to electromagnetic radiation in the infrared-visible-ultraviolet region of the spectrum.

These sensors measure many quantities including light intensity, color, displacement, flow, and temperature. They are also useful in imaging and fiber-optic communications.

They are divided into two categories: ***photon detectors, and thermal detectors***

Photon detectors are sensitive to photons with energies greater than a given intrinsic energy of the detector material. Their sensitivity increases linearly with increasing the wavelength up to a cutoff wavelength.

Types of photon detectors are: Photoconductive detectors, photovoltaic detectors, photodiodes, and photoemissive devices.

Photoconductive detectors are based on the photoconductive effect which is the most widely used. Using this effect, incident photons with energy greater than a certain level cause electrons to produce hole-electron pairs resulting in an increase in the conductivity of an intrinsic semiconductor.

Photovoltaic detectors are known for their ability to convert sunlight into electrical energy but they are also used as optical sensors. These detectors use a junction of two dissimilar materials. Potential barrier is created at this junction where a carrier flow is excited by incident photons. This results from hole-electron pairs created near the junction. These detectors do need an external power source, and are useful for low-light applications.

Photodiodes have a linear relationship between their photocurrent output and the incident energy. They are based on photon-induced electron-hole production. Schottky photodiodes are fast, highly-sensitive, expanded spectral detectors. They also have small dark currents and relatively little noise. They, however, are not recommended for high temperatures or high light-level applications.

Photoemissive devices and especially the photomultiplier tubes (PMTs) are the most sensitive photodetectors. These devices can be made very sensitive, and the relationship between photocurrent and illumination is linear over a wide range. PMTs consist of a photocathode coated with an alkali metal which emits photoelectrons when high energy photons strike it. PMTs multiply the photocurrent using several dynodes at successively higher voltage.

Thermal detectors are the second category of photon detectors. They are also used as temperature detectors. Examples of the types of thermal detectors are Bolometers, thermopiles, and pyroelectric sensors.

Bolometers consist of two matched elements in a bridge circuit. One of the elements is coated with an absorptive coating and illuminated, while the other element is shielded from radiation and isolated thermally in a heat sink. The elements are made of a similar material as thermistors. The incident radiation changes the resistance of the first element which in turn changes the balance in the bridge circuit.

Thermopiles are constructed from thermocouple connected in series in which alternate junctions are coated and exposed or shielded and thermally isolated. Semiconductor elements are used for high sensitivity.

Pyroelectric sensors use pyroelectric crystals which go through a spontaneous change in polarization when the temperature changes. These sensors are capacitive in nature and like the piezoelectric sensor have no dc response. The rise time of these sensors can be made as low as 1 pico second which make them superior at very high frequencies over other sensors.

For examples of some common sensor systems and their interface to the IBM PC see the reference text pages 244-248.

1.3.3 Displacement Sensors

Displacement sensors are very important for many industrial applications. In this section, potentiometric displacement transducers, strain gauge transducers, and piezoelectric sensors are discussed.

The potentiometric displacement transducer is one of the simplest and most efficient displacement transducers. It consists of a sliding contact which is connected to the moving object while the rest of the potentiometer is attached to a stationary object. Both transactional and angular displacements can be measured. The resolution of transactional displacement can be made as high as 0.2%, with linearity of 0.4%.

Strain gauge transducers are based on a mechanical strain which causes a displacement resulting in a change in resistance. Strain gauges can be made from metal or semiconductor material, or from an elastic material. Metal or semiconductor strain gauges measure very small displacements less than 20 micrometer, while elastic strain gauges measure large displacements.

Strain gauges are classified as unbonded and bonded gauges. The unbonded gauge has one fixed end and one movable end connected to the force member which can cause the mechanical strain. The bonded gauge is entirely attached to the member whose strain is to be measured.

Semiconductor mate-‘ıl, usually silicon, is used for strain gauges. The strain gauge properties can be optimized for a specific application by controlling the amount and type of dopant in ‘he silicon. Bonded gauges can be made by slicing sections from specially processed silicon crystals doped as n or p types. These gauges have high gauge factor, or strain sensitivity, however it suffers from high temperature sensitivity, nonlinearity, and mounting difficulties.

The effect of temperature in the use of strain gauges is an important source of error. One of the best ways to achieve temperature compensation is to use a four-gauge balanced Whetstone bridge.

Strain gauges are widely used in force measurement and pressure measurement. An example of a pressure transducer is the National Semiconductor LX1701G. It is a fully signal-conditioned transducer with an output voltage of 10 V. The transducer can be thermostatically controlled thus allowing only a very small change in the output voltage over large temperature variations.

Elastic-Resistance Strain Gauges are used extensively in biomedical applications, especially in cardiovascular and respiratory dimensional determination. A typical gauge would consist of a rubber tube (0.5 mm ID, 2 mm OD) from 30 to 250 mm long,filled mercury or with an electrolyte or conductive paste. The ends of the tube are sealed with electrodes (amalgamated copper, silver, or platinum). As the tube stretches, the diameter of the tube decreases and the unstretched length increases, causing unstretched resistance to increase:

Piezoelectric Sensors are based on the piezoelectric effect. The piezoelectric effect arises when a crystal generates a charge or voltage as a result of mechanical stress. In this case the distortion due to stress in the crystal lattice produces a charge reorientation which causes the relative displacement of positive and negative charges to the opposite outer surface of the crystal. Piezoelectric properties are found in materials such as synthetic ceramics, and vinylidene fluoride polymers.

Piezoelectric transducers can be used to measure force, displacement, pressure, and acceleration. The pressure transducer normally uses a diaphragm to react with pressure and produce the stress on the piezoelectric crystal. It has the ability to operate over a wide temperature range with relatively small errors. Piezoelectric force transducers normally use the strain or the deflection of the sensing element. Their application is mostly in dynamic force measurements although they can also be used in measurements of quasi-static response when they are used in conjunction with charge amplifiers.

Piezoelectric accelerometers use the force on a crystal to measure acceleration. The crystal may be bonded to the mass which also acts as the elastic member. Another design uses an annular crystal bonded to a center post on its inside surface and to an annular mass on its outside surface. The upward or downward deflection of the mass causes shear stresses across the thickness of the crystal. Typical compression designs use a stacked arrangement to increase the low output of quartz crystals. One transducer uses seven crystals which are stacked and connected for output multiplications. Piezoelectric accelerometers are widely used in vibration measurements because of its ruggedness, reliability, and simplicity.

1.3.4 Flow Sensors

In this section we briefly discuss thermal, mechanical, electromagnetic, and ultrasonic methods to measure fluid flow.

Thermal flowmeters are based on an inserted heat element in the flow path of the fluid. The rate of heat transfer from the element to the fluid depends on the temperature difference between the element and the fluid, the speed of the fluid, and the flow profile. The flow speed can be measured by measuring the heat lost from the heating element or the fluid temperature change due to heat infusion. The later method however has many potential inaccuracy and therefore is impractical.

Mechanical flowmeters are based on having the fluid physically deflect a measuring element such as a turbine or propeller. The rotation speed of a turbine or propeller is easily measured with an optical or magnetic sensor which outputs one pulse for each rotation. If the output pulses are TTL compatible they can be connected directly to the PC. The instantaneous flow can be found by taking the reciprocal of two consecutive pulses, and the average flow can be measured by counting the number of pulses in a given time interval. Rotating flowmeters are quite accurate when properly calibrated and maintained. Periodical maintenance is essential due to tear and wear.

Electromagnetic flowmeters are used to measure the average flow of a conductive liquid. They are based on the effect that a conductor perpendicular to a magnetic field will induce a voltage across the conductor linearly proportional to the liquid velocity. Many sources of error exist in these flowmeters. The error in measuring the blood flow in biological studies, for example, may easily exceed 10%

Ultrasonic Flowmeters use piezoelectric material which converts power from electric to acoustic form. In measuring flow, they rely on one of two physical principles. The first is based on the fact that the effective velocity of sound in a moving medium depends on the velocity of the medium itself. The second principle is based on the Doppler shift which is a change in frequency that occurs when an ultrasonic wave is scattered by a moving medium. These flowmeters have low noise and are highly efficient. They are used for measuring flow in many industrial and biological applications.

For a detailed example on interfacing a Parks Electronics Laboratory Model 810 nondirectional continuous-wave ultrasonic Doppler flowmeter to the IBM PC, see the reference text pages 326 to 331.

1.4 SIGNAL CONDITIONING DEVICES (reference text: [GATES&BECKER 89])

In this section, signal conditioning devices used with the above described sensing devices are discussed. These are used for signal filtering, level and gain conversion, isolation, noise reduction, and protection. In general, devices needed to suppress the effect of noise on the often very small signals measured by the various types of transducers will be discussed.

1.4.1 Noise Sources

We first start by discussing the sources of noise. Noise sources can be either internal or external, i.e, they are either internal to the devices or external due to the connections and coupling of devices.

Internal noise sources are, for example, passive or active thermal noise, random shot noise, and A/D quantization noise. External noise sources are mainly introduced from nearby instruments, motors, fluorescent lighting, and other external devices.

Passive thermal noise is caused by the effect of temperature on passive circuits elements such as resistors and capacitors. Resistors however, are the main consideration when designing low noise interface circuits using passive circuit components. This thermal noise is generated by the currents produced by electrons colliding with the resistive material. The average thermal noise generated by a 10 Mohm resistor in room temperature in a 10 KHZ bandwidth is about 40 micro volts. In an experiment using a 16 bit A/D converter in a 5 volt range, the resistor noise voltage can approach 1 bit of uncertainty in the A/D conversion.

Random shot (or Shottky) noise is also generated by a resistor subject to dc current because of the discrete charge of electrons and the collisions of these electrons inside the resistor. This noise can also be a factor in sensitive measurements.

The thermal noise voltage and the shot noise voltage can be minimized by controlling the temperature and dc current flowing through a resistor. This can be done by choosing resistors that minimize the dc currents to a shot noise level below the thermal noise level.

Active thermal noise sources are the semiconductor devices, such as transistors and diodes, used in interfacing circuits, such as amplifiers and A/D converters, which are sensitive to a small variation in temperature. These temperature-sensitive devices were actually used in the design of temperature sensors as mentioned in the previous section. Many common op-amps composed of semiconductor devices are not designed for ultra-low noise measurements. It is possible to compensate for the drift noise generated from these devices by incorporating temperature sensing and correction circuitry. Precision op-amps and high precision A/D converters (14-16 bits) can provide added compensation for the drift noise internally.

A/D quantization noise is another noise source related to the conversion of an analog signal to a digital signal. For example, a 12 bit A/D converter can digitize a 10 volt analog signal with a resolution of 5 Mv. If the analog signal is a dc signal then the digitized output may jump between two integer values around the dc value causing a noise in the measurements. The quantization noise can be reduced by using a higher resolution A/D converter or by filtering out the quantization noise later during data analysis. This later case will be elaborated on further when hardware and software filtering techniques are discussed in this section.

Next we discuss external noise. External noise can be divided into two categories, namely, electrical noise and magnetic noise. Electrical noise is generated from currents produced due to coupling external electric fields and wiring in the system under design. Magnetic noise refers to currents produced when wires are placed in a large changing or alternating magnetic fields such as those found in electric motors.

The effect of external noise sources can be reduced by shielding, avoiding ground loops, using guards and optical isolation, and the proper selection of cables and their connections. Each of these techniques are briefly discussed next.

1.4.2 Shielding and Isolation techniques

Shielding is used as a simple solution for reducing the effect of capacitive coupling between power lines electric fields and conductors in our circuits. For example, a grounded metal shield can be placed around the conductor driving an op-amp. Line frequency noise (50 or 60 HZ) will flow through the shield to the signal ground rather than the op-amp input. The wire shields should be grounded in locations of greatest encountered interference affecting our analog signals. This is usually at the transducer end where signals are very sensitive. The shield may not eliminate the effect of capacitive coupling but it is known to reduce this noise signal by a factor of 100 to 1000.

Avoiding ground loops is the second technique to be discussed in noise reduction. Ground is not an absolute value but can actually vary significantly from one place to another. When the analog signal transducer and the A/D converter are connected to two separate grounds, a difference in grounds potential usually exist and is called a common mode voltage (CMV). A ground loop is said to be formed with the existence of a CMV. This CMV can be a major source of error in the measurement detected by the A/D converter.

There are several ways to avoid or reduce the effect of ground loops. The best way is to minimize the distance between the transducer and the A/D converter. The second way is to connect the grounds by a heavy conductor (a ground strap) cable. Since this cable has a very small impedance, the voltage drop across the cable, which constitutes the CMV, will be minimized.

The effect of alternating magnetic fields can produce substantial alternating currents in a long ground conductor cable. The effect is minimized using guards and optical isolation. A guard is a shield that is connected directly to the CMV between the signal source and the A/D converter. Any sizable CMV is effectively shielded by the guard connection. A quality measure, called the common mode rejection ratio (CMRR) is often used to specify the effectiveness of a technique in reducing the effect of CMV. The CMRR is defined as the ratio between the CMV and the signal voltage received at the input of the A/D converter. It is possible to achieve a CMRR of 10^6.

Optical isolation is one of the most effective techniques in improving the CMRR. An optoelectronic isolator is composed of a light emitting diode (LED) connected to the analog transducer circuit, and a photo-transistor connected to the A/D converter circuit. The LED and the photo-transistor are mounted closely together in an IC package. The light from the LED illuminates the base of the transistor which controls the current conducting through the transistor. Since there is no electrical connection between the transducer circuit and the A/D circuit, a high CMMR is achieved. This technique is particularly useful in large signal applications.

1.4.3 Filters

Next we discuss techniques used to filter out noise using analog and digital filters. Filters can be implemented in either hardware or software. Analog filters (such as low-pass filter, high-pass filter, etc.) can be implemented in software, however they are most commonly implemented in hardware. Digital filters can be implemented in hardware, although they are most commonly implemented in software.

Analog hardware low-pass filters are used to eliminate the harmonic distortions which are high frequency noise generated by analog transducers such as photodiodes and piezoelectric sensors. An aliasing error is produced in the data samples when an A/D is used to sample the signal obtained from these transducers (see designated reference for this section, pp 73-75, for an example).

A software filter, which is considered to be on of the simplest noise reduction technique is the *bunching* filter. This filter, which is a simple type of low-pass filter, is based on replacing each n consecutive data points by a single point which is the average value of these n points. A bunching filter reduces the effect of randomly distributed noise with the advantage of being very rapid so that it can be done in real-time. Bunching rely on a high data sampling rate from the source to reduce the effect of loosing information when relating n points with a single point (the sampling rate after bunching should be at least twice the highest frequency component in the input signal which is known as the Nyquist rate).

The moving average software filter is an averaging technique with no reduction in the number of data points. This smoothing technique can also use a set of smoothing functions and therefore is much more general than the bunching filter described above. Other filtering techniques use the waiting functions with the same effect as fitting a least-squares line through the data points. These filters, however, are more computationally intensive than the above bunching filter.

Digital filtering using fast fourier transform (FFT) is another effective form of noise reduction. A software filter using the FFT provides ideal frequency filtering of the acquired data. A large number of data samples must be acquired first, however, before filtering based on the FFT can be performed. Implementing the filter in software is very computational intensive especially for two-dimension and three dimension FFT. Special hardware boards for signal processing is usually used and is much more effective than a pure software implementation.

1.4.4 Gain conversion and Level Shifting

Signal conditioning circuitry include circuits for two other important functions, namely, offset adjustments, and gain adjustments. The offset adjustment function sets the input signal minimum voltage level to be similar to that required by the A/D converter. The gain adjustment is set so as to allow the maximum input signal to derive the A/D input to its maximum converted value. For an example of such a circuit see [LAWRENCE & MAUCH 87] pages 314-315.

1.5 ACTUATORS (designated reference [LAWRENCE&MAUCH 87])

The output of a real time system is usually directed to devices used to control another system or a process. Such devices are called actuators.

An actuator may often consist of a transducer and an amplifier. The transducer converts the electric energy to the required control output (e.g. an electric motor converts electrical energy to mechanical motion). The amplifier amplifies the computer interface output signal to the level needed to drive the transducer.

Actuators can either have only two states or a continuous state. A two state actuator is either ON when the electric energy is applied, or OFF when no energy is applied. A continuous state actuator has a continuous output variable over a range of values.

1.5.1 Outputs Using Two-State Actuators

A two state actuator is designed using a switch. No D/A conversion of the computer output signal is necessary since the output signal can be used to control the states of the switch. These switches can be either electrically isolated or non-electrically isolated.

Examples of non-electrically isolated switches are integrated circuits containing transistor switches such as the Sprague ULN2069B, the discrete power transistors, or MOSFETs. The UNL2069B consists of four transistor switches driven by TTL or CMOS signals and therefore are considered to be low power switches. The discrete power transistors (such as the TIP 122) or MOSFETs (such as the RCA RFP10N12L), on the other hand, can carry 5 to 10 Amp current and block 100 to 120 V and therefore are used for switching medium power actuators.

Electrically isolated switches, used for high power applications, are *electromechanical relays (EMRs) and solid-state relays (SSRs).*

EMRs used to be the most common switches used in control applications. An EMR consists of a relay coil, an armature, and output two contacts. When an input excitation current flows through the coil, the armature is pulled away from the normally closed output contact (which transmits the input power to the output) to the normally opened output contact in which no power is transmitted.

The advantages of EMRs are as follows: can handle both direct or alternating currents; available in a wide range from small relays used to switch millivolt signals to large conductors used to switch hundreds of kilowatts of power; can handle momentary overloads without damage.

The disadvantages of EMRs are as follows: large switching time (switch in milliseconds) which might cause large delays; large electrical noise are generated due to contact bounce which can affect the computer or other sensor systems operations.

SSRs were developed to overcome the deficiencies of EMRs and are now being used instead of EMRs in many applications. They consist of a control input which is coupled optically or inductively to a solid state power switching device.

An SSR can either be designed to switch a direct current or an alternating current but not both. When designed for switching a direct current, the switching device is either a power transistor or a power MOSFET. For alternating currents, the switching device is a *triac*. The triac is a member of a class of semiconductor power switches called *thyristors*. When a thyristor is switched on (i.e, fired) it stays on as long as the current flows in the switch and switch off when the current goes to zero. They can handle considerably more power than power transistors or power MOSFETs.

The advantages of SSRs are as follows: low noise since they are usually designed to switch at the zero-crossings of the alternating voltage (hence eliminating the noise caused by the rapid rise of current if the switch is closed while the voltage is nonzero; the control inputs can be directly obtained from digital logic circuits hence they can be easily interfaced to computers.

1.5.2 Outputs Using Continuous Actuators

Continuous actuators require the use of a D/A converter for the computer output interface to produce a voltage proportional to the value of the binary data. This voltage is applied to the input of a linear amplifier which then amplifies the signal to a proper level in order to be applied to the actuator.

Power operational amplifiers are needed since the output of A/D converters is limited to milliwatts. These op amps produce power levels up to a few hundred watts. Examples of such power op amps are the Microtechnology PAO3 and the National LM675.

For applications requiring higher power ratings, servo amplifiers or programmable power supplies are often used. Servo amplifiers are used in order to supply power to electromechanical transducers such as motors or electrically controlled valves. Programmable power supplies are power supplies whose outputs are controlled by analog or digital signals. They have higher accuracy and stability ratings, at the expense of a larger response time as compared to servo amplifiers. The maximum rate of change of a programmable power supply is in the order of 1 V/ms.

To control the power supplied to the load, the technique of *pulse-width modulation*

(PWM) is used in many amplifiers. PWM amplifiers contain solid-state switches that control the amount of power applied to the load. These switches transmit power to the load by turning on and off at a constant frequency but with a variable duty cycle, i.e., with a changing pulse width, the larger the pulse the larger the amount of power transmitted. PWM amplifiers have a low power dissipation factor and are easy to interface to a microprocessor. In this case, the microprocessor can be used to produce a pulse-width modulated signal to control the switching, and a continuous actuator can be controlled without the need of a D/A converter and a separate analog amplifier. See the designated reference (pages 336-337) for examples.

1.5.3 Examples of Actuators

In this section, some examples of actuators are discussed. These are the stepping motors and servomotors.

1.5.3.1 The stepping motor

The stepping motor produces incremental motion from electric pulses applied to its windings. A single step is taken due an excitation of the stator windings and the rotor translates to a new position. The excitation through the stator flux and the rate of increase of the flux determine the maximum kinetic energy input to the rotor. The load friction is in the form of a damping. The system inertia consists of both the rotor and load inertias. The acceleration performance of a steeping motor is a function of the driver, load friction, inertia, ramp time, starting frequency, and final frequency. A digital positioning system can be used to generate motion pulses having a linear velocity ramp such that the last pulse of motion and zero velocity occur simultaneously.

Stepping motors can be either permanent magnet (PM) or variable reluctance (VR) motors. The PM motor uses permanent magnet for the rotor assembly. The stator has a number of wound poles, and each pole may have several teeth for flux distribution. When the pattern of winding energization is fixed, for a given excitation, a series of equilibrium points are generated around the motor and the rotor will move to the nearest of these and remain there. When the windings are excited in sequence, the rotor will follow the changing equilibrium points and rotate in response to the changing pattern. When the excitation is removed, the use of a permanent magnet causes a small residual holding torque in the motor.

The VR stepping motor also has a number of wound poles, and the rotor is cylindrical with teeth that are related to the stator poles. This relationship is a function of the step angel required. When a current flows through the windings, a torque is developed which tends to move the rotor to a stable position of minimum magnetic reluctance. This minimum reluctance point changes, when a different set of windings is energized, causing the rotor to move to a new position. In a VR motor, the rotor has a very small residual magnetism than a PM motors, thus there is no torque when the stator is not energized.

Microprocessors are usually used to control the movements of stepping motors. Some hardwired logic are also needed to advance the motor by a step when an output appears on a state generator. The logic sequence for state generation can also be stored in memory and a state vector (a byte of information) is sent out one at a time. The control strategies can be done in two modes, a constant speed mode in which the motor is moved the sequence of steps in a constant rate, or an acceleration-deceleration mode in which progressively decreasing or increasing time delays are used between the steps to increase or decrease the stepping rate.

The advantages of stepping motors are as follows: they are easily controlled by microprocessors since windings currents are only needed to be switched on or off; the speed of rotation can be controlled by controlling the time between successive steps; the direction of rotation can be controlled by the winding excitation sequence; the rotor position can also be controlled and monitored by counting the number of steps; and finally if the motor is left energized in the same state once a target position is reached, the rotor will resist movement.

For examples of interfacing stepping motors to microprocessors see the designed reference for this section pages 337-346, and also see [HOEDESKI 85] pages 201-221.

1.5.3.2 DC Servo Motors

DC servo motors are used in applications where the position and velocity of the actuator are to be controlled. However, unlike stepping motors, it is not possible to control the rotor velocity or position of a servo motor without the use of feedback sensors. Therefore, dc servo motors often come equipped with feedback sensors such as tachometers (for velocity feedback) and shaft encoders (for position feedback).

The characteristics of dc servomotors are specified by the manufacturer in such detail as to allow a system designer to determine the motor suitability for a particular motion control application. The specifications provided include the peak current, the armature power dissipation, and the maximum speed. Once it is verified that power dissipation for the application at hand is safely below the maximum limit, and the maximum current and velocity are less than the peak current and velocity ratings of the motor, the voltage and the current ratings of the servo amplifier that will power the motor is then determined.

The interface to the microprocessor is then designed according to one of the following three cases: The first case is using an analog servo loop in which the microprocessor now positions data through a D/A converter to an analog feedback positioning control loop; the second case uses a mixed digital/analog servo control loop in which position control is done using a digital control circuit; the third case uses a digital servo controller board which communicates with the microprocessor to produce the armature control current supplied to the servo amplifier. For more details see the designated reference pages 352-354.

1.6 DATA ACQUISITION BOARDS

[reference Text: The Handbook of Personal computer Instrumentation, Intelligent instrumentation]

1.6.1 Data Acquisition and Control Overview

Data acquisition is the collecting of information that describes a given situation when a given condition is satisfied. This condition can be defined by a uniform time base or an event. "Real-time" systems are characterized by their ability to perform a given data acquisition and/or control task within set time period. The speed of the real-time system is dependent on the accuracy and speed of the given application.

In the past, data collection and control was done by technicians, engineers, physicists, chemists, or others involved in research, testing, development, production, quality control, management, process control, etc. Today, the emphasis is on getting machines to do these jobs. The motive is productivity, speed, accuracy, dependability, reliability, and cost.

In the past, an acceptable automated data logging system consisted of strip chart recorders, printers, and tape recorders. When monitoring was not enough, programmable controllers were added to match the desired task. However as technology increased, the data loggers and programmable controllers were not sufficient to meet the desired tasks. This was due to the narrow range of functions supported by such devices. Today this problem has been solved. Data acquisition equipment based on the digital computer has replaced the automated data loggers and programmable controllers of the past.

Modern digital computers offer high speed, flexibility, adaptability, consistency, reliability, and mass memory. These features allow for extensive capabilities for mathematics, analysis, storage, display, report generation, control, and communications. However many real-world signals such as temperature, pressure, and speed cannot be read directly into the computer because of their analog nature. Therefore, a device must be used to translate these signals into a digital representation that the computer can understand.

Data acquisition and control devices (DA&C) translate real-world signals into a format that digital computer can accept. Also, DA&C's can generate analog signals and other wave forms from instructions issued by the host computer and send them back into the real world. The most accessible host for DA&C's, is the personal computer (PC).

The PC's presence can be seen in a great many areas of applications. These include:

(1) Laboratory data collection and automation;

(2) Medical instrumentation and patient monitoring;

(3) Automatic test equipment (ATE) for incoming inspection, life test, burn-in, production test, and final test;

(4) Industrial monitoring and control; and

(5) Environmental and utility management.

1.6.2 DA&C Systems and How They Connect to the PC

There are two different means of connecting a DA&C system to a host computer. The first is a direct connection to the PC bus. The second is a connection through an external data line such as a RS-232, RS-422, or IEEE-488. Each method has its advantages and disadvantages.

1.6.2.1 External Bus Products

There are several advantages with external bus products, which includes:

(1) Virtually any size system can be configured;

(2) The DA&C system can be placed remotely from the host computer, thus being closer to the field signals;

(3) The DA&C system can relieve the host computer of some of the data-collecting tasks;

(4) The DA&C system can be interfaced to virtually any type of computers.

The use of external bus systems allows for the construction of distributed systems. Thus, a large number of parameters can be monitored or controlled even though the location of each signal is far from each other and the host PC. This type of capability can greatly improve productivity and reduce overall system cost.

1.6.2.2 Internal Bus Products

The advantages of internal bus connections include:

(1) High speed;

(2) Low cost; and

(3) Smaller size.

The cost is reduced with this kind of DA&C system because it does not require a separate enclosure or a power supply. When the data acquisition hardware resides inside the PC, size is reduced and space is utilized more efficiently. Higher speed is obtained by eliminating the slow protocols of the communications channel being used. For example, the RS-232 protocol will allow for only 20 samples per second. In comparison, some direct connect systems allow for more than 1 million samples per second.

1.6.2.2.1 Types of Internal Bus Products

There are two major types of internal bus products. They are distinguished by the way in which the Input/Output channels are configured. Both types are board-level systems that make a direct connection to the PC bus. The first type of boards have a fixed arrangement. While the second type is based on a modular approach.

1.6.2.2.1.1 Fixed Configuration Internal Bus Products

A fixed configuration system is a board that retains the original configuration at all times. These boards can not be expanded for modification for future needs. Limitations of this type of systems include lack of channel expansion capability and the inability to add functions not originally purchased. The advantage is that the cost is usually the lowest.

Often, the use of fixed configuration boards requires great compromises. The user may not be able to achieve the required number of channels or must buy functions that are unneeded. It is inevitable that a mismatch between the available channels and the actual requirements will exist. To solve this problem, some fixed configuration boards allow for selected types of channel expansion via external add-on boards or boxes.

1.6.2.2.1.2 Modular Internal Bus Products

Modular board systems are far more effective in satisfying a user's needs. These boards can be easily tailored to the precise needs of any system. The addition of new components can be as simple as plugging them into the existing card.

1.6.3 Data Conversion Principles

As mentioned earlier, most real-world systems do not fit the format of the digital computer. It is the job of the data acquisition system to perform all conversions necessary to allow the computer to understand a given signal. To do this, the data acquisition system uses components such as analog-to-digital (A/D) converters, multiplexers, sample/holds, amplifiers, counter/timers, and other more specialized functions. The most important function of the data acquisitions system is that it brings together these functions in a compatible, integrated system. Given the software to drive the system, a user does not have to be intimately familiar with the internal details of the conversions.

1.6.3.1 ANALOG INPUT SYSTEMS

The function of the analog input system is to convert the analog signals into a corresponding digital format. It is the analog-to-digital converter (A/D) that performs this task. In addition to the A/D, other components such as amplifiers, sample/holds, multiplexers, and signal conditioning elements may be used to gain optimum performance.

1.6.3.1.1 ANALOG-TO-DIGITAL CONVERTERS

Today there are many different types of A/D convertors. From this vast group, three tend to stand out as the most widely used: successive approximation, integrating, and parallel (flash) converters.

1.6.3.1.1.1 SUCCESSIVE APPROXIMATION A/D CONVERTORS

The successive approximation A/D is normally used for speeds above 100 samples/second. It has a maximum speed capability of 100K samples/second. The conversion is done by using binary weighted guesses and comparing them to the actual input signal until a match is achieved. It is essential that the input signal remain constant during the course of the successive comparisons or very significant errors can result. For this reason, a sample/hold circuit is necessary (see section 1.6.3.1.4 for sample/hold description).

1.6.3.1.1.2 INTEGRATING A/D CONVERTERS

When high speed is not required, an integrating A/D converter may be used. This converter can give 12-, 14-, or even 16-bit resolution at low cost. Sampling speed is typically on the order of 3 to 50 conversions per second. The conversion is performed by averaging any input signal variations during the conversion cycle. This technique inherently filters input noise. Integrating A/D converters generally have better linearity and overall accuracy than other A/D converter.

1.6.3.1.1.3 PARALLEL (FLASH) A/D CONVERTORS

Flash converters are the fastest and most expensive form of the A/D. The complexity of this device generally limits these devices to low-resolution (8 bits or less). With this converter, the search to determine if each bit's digital value is a 1 or 0 is accomplished in parallel and the time of conversion is determined by the delay through the circuitry.

1.6.3.1.1.4 A/D ACCURACY AND RESOLUTION

Accuracy is an important measure of an analog input system. It defines the total error in any particular measurement. For example, a system which is specified as 0.05% accurate of full scale on a 10 volt range, would have a worst-case error of 5 millivolts. A system with an accuracy of 0.1% on the ±10 mV range would have an error of 20 microvolts. In assessing the value of a data acquisition system, the accuracy specification requires careful scrutiny.

The resolution of an A/D will determine range of a detectable signal. A 12-bit system provides a resolution of one part in 4096 or approximately 0.025% of full scale. 16 bits corresponds to one part in 65,536 or approximately 0.0015% of full scale. Therefore, resolution not only determines the range, but it also limits the overall system accuracy.

1.6.3.1.2 AMPLIFIERS

For an A/D converter to perform at its best, the input signal must be high. In many systems, an amplifier is used to boost possible low-level signals to the desired amplitude. These amplifiers come in two forms. The first form is a programmable gain amplifier. This type has several gain settings that are controlled via software. The second type is the manual adjustment amplifier. This amplifier is set through the selection of a resistor or the setting of a jumper.

The use of an amplifier can increase resolution and accuracy greatly. For example, amplifying a low-level signal by 10 or 100 increases the effective resolution by more than 2 and 6 bits respectively. Therefore, a 12-bit converter has the dynamic range of 15 to 18 bits.

1.6.3.1.3 MULTIPLEXERS

Multiplexers are often used in acquisition boards to tie several channels to a single amplifier and A/D. This method greatly reduces the cost of the systems. Software can be used to select which channel will be processed. Since the amplifier and A/D are being shared, the overall speed of the system will be reduced. To a first approximation, the rated speed of the amplifier and A/D will be divided by the number of input channels.

1.6.3.1.4 SAMPLE/HOLD

A sample/hold will capture the current value of an analog input signal and prevent it from varying. This is needed for successive approximation A/D converters. The captured level is held constant during the duration of the conversion. This allows for an accurate conversion of high-frequency signals.

1.6.3.2 SIGNAL CONDITIONING

Signal conditioning is the pre-processing of input signals. It is divided into two categories. Active signal conditioning can include amplification and isolation, while passive signal conditioning includes voltage division, surge suppression, current-to-voltage conversion and filtering.

1.6.3.3 SINGLE-ENDED V.S. DIFFERENTIAL SIGNALS

Analog signals can be configured as either single-ended or differential inputs. Single-ended inputs all share a common return or ground line. Only the high ends of the signals are connected to the multiplexer. The low ends return to the amplifier through the systems ground. This arrangement works fine as long as the potential difference between the line and ground is relatively small. The main advantage of single-ended inputs is the low per channel cost. Only one multiplexer switch is required to handle each input channel.

In a differential connection, both the inverting and non-inverting terminals of the amplifier are connected to the input signal. Any ground-loop-induced voltage appears as a common-mode signal and is rejected by the differential properties of the amplifier. A drawback to this connection is the need for two multiplexer switches per channel. Thus, a 32-channel single-ended system can only support 16 channels of differential connection. Also, a standard op-amp can not be used for differential connections. An instrumentation type amplifier is required for differential inputs.

1.6.3.4 ANALOG OUTPUTS

In many applications analog output signals are needed to drive a variety of tasks. These signals are may be used for chart recorders, to provide feedback, and to initiate various functions. Common output ranges include ±5V, ±10V, 0-10V, and 4-20mA.

Most D/A converters can supply up to 5 or 10mA of load current. This is normally not a limitation, because the majority of the applications call for driving high impedances. When large loads such as positioners, valves, lamps, and motors are controlled, power amplifiers or current boosters are required. Most DA&C systems do not come standard with high-power analog drivers.

1.6.4 SYNOPSIS OF CURRENTLY AVAILABLE DATA ACQUISITION BOARDS

1.6.4.1 ADAC CORPORATION -- MODELS 5525MF AND 5550MF

[reference: Models 5525MF, 5550MF data sheet]

These boards are PC bus plug-in multifunction I/O boards. They feature 16 analog inputs with 12-bit resolution and a ±10V input range. The boards also feature software programmable amplifiers. The 5525MF has a maximum throughput of 40kHz. While the 5550MF has a maximum throughput of 60kHz. Both boards have 16 digital I/O lines, programmable timers, and pulse and frequency measurements. Also, two analog output can be added along with signal conditioning.

To drive the boards, a set of callable subroutines for MS/PC-DOS are available. The subroutines allow access to all I/O functions. The languages supported by the drivers are:

FORTRAN, Pascal, IBM BASICA, GW-BASIC, Quick BASIC, C, and Quick C.

1.6.4.2 QUATECH -- DAQ-12

[reference: QUATECH 1993 Data Acquisition/communications Handbook]

The DAQ-12 is a 12-bit data acquisition system with a maximum throughput of 200kHz. It has 8 differential A/D channels or 16 single ended A/D input channels. It comes standard with 2 12-bit D/A output channels. The amplifier has a software programmable gain of 1, 2, 4, 8, 10, 100, 500 or 1/2, 1, 2, 4, 5, 50, 250. This system is interfaced directly to the PC bus via a standard card slot.

The DAQ-12 has a set of software drivers that allow access to all functions of the board. The driver package includes interfaces to BASIC, C, FORTRAN, and TURBO Pascal.

1.6.4.3 INDUSTRIAL COMPUTER SOURCE -- LC16-125

[reference: 1993 I/O SOURCE-BOOK]

The LC16-125 is a multifunction I/O card for PC's. This card interface directly to the bus via a standard ISA slot. This card features 16 single-ended or 8 differential analog input with 12-bit resolution and 125kHz sampling rate. Also, included are 2 12-bit D/A convertors. The is also a programmable gain amplifier used along with the A/D.

Software support is available that allow for customer programming of the board. The library supports Microsoft C and Borland Turbo Pascal.

1.6.4.4 INDUSTRIAL COMPUTER SOURCE -- AI08G-P

[reference: 1993 I/O SOURCE-BOOK]

The AI08G-P is a ISA PC standard card. It has 8 differential A/D inputs with 12-bit accuracy and ±10V range. They use a successive approximation converter that gives a speed of up to 20,000 samples per second.

The board has the capabilities of being directly accessed through assembly language or the supplied drivers. The drivers support BASICA and GW-BASICA.

1.6.4.5 INDUSTRIAL COMPUTER SOURCE -- AI016-P

[reference: 1993 I/O SOURCE-BOOK]

The AI016-P is an ISA PC standard card. It has 16 single-ended or 8 differential analog inputs. It samples at a rate of 50,000 and 100,000 samples/second with 12-bit resolution. It comes standard with 2 analog outputs. The gain on the amplifier is manually switched on the card. Also, additional gain ranges may be achieved by adding a gain resistor to the board.

Software is available that runs under windows and a Visual BASIC library for windows that allow access to the boards functions.

1.6.4.6 TRANSERA -- MODEL 410

[reference: TransEra model 410 data sheet]

The model 410 is a low power, high performance analog, digital, counter, and timer board for PC's. The board fits directly into expansion slots in the computer. It contains a 13-bit sampling A/D convertor with up to 16 single ended or 8 differential analog inputs. Also, there is a programmable gain amplifier with range of 1, 2, 4, and 8 and a pseudo-simultaneous sample and hold.

Software support includes complied C drivers for use with TransEra HTBasic, Microsoft C, and Borland C.

1.6.4.7 CYBORG -- LOGGERNAUT II MODEL 910

[reference: LOGGERNAUT II MODEL 910 data sheet]

The LOGGERNAUT II model 910 is an external bus device. It interfaces to the PC via a RS-232C port. It has 16 single-ended and 8 differential analog inputs. It is expandable up to 64 channels. It has the capability of direct sensor inputs of J, K, and T thermocouple, RTD's, 0-100mV, 0-1V, 0-5V and 4-20mA. The channels can be configured in blocks of four instead of all 16 the same. It also has automatic cold junction compensation and signal linearization for thermocouple inputs.

The system comes with ready to use software that can run the entire acquisition system. Also, software drivers are available for costumer programming in BASIC, FORTRAN, and C.

1.6.4.8 MICROSTAR LABORATORIES -- DAP 3200e

[reference: DAP 3200e data sheet]

The DAP 3200e can acquire up to 512 analog inputs and 128 digital inputs, can process the acquired data, and can update up to 66 analog outputs and 128 digital output. On board are DC/DC converters, D/A converters, A/D converters, programmable gain amplifiers, and digital I/O. The card has an overall mixed analog and digital sampling rate at over 4 megasamples per second. These capabilities are obtained through the use of an Intel 80486SX microprocessor and 4MB of RAM on the card. Also, the card has its own multitasking real-time operating system on-board. This allows for the host computer to be freed up to do other tasks.

The on-board multitasking real-time operating system has over 100 standard commands for on-board processing, including Spectral Analysis. These commands configure the DAP, smooth data, condition sensor data, wait for or generate triggers, respond to alarms, and format output.

CHAPTER II. HARDWARE DEVELOPMENT STEPS AND TOOLS

(reference text: [PEATMAN 88])

2.1 HARDWARE DEVELOPMENT STEPS

Hardware development follows similar steps as those used in software development life-cycles described in the software engineering course, the first course in this sequence, for developing the application software. The hardware development model, however, is centered around prototying in which a prototype is built and used to test and verify the design decisions. Simulations can be used to simulate the microcontroller's operation and its I/O interactions to debug the low-level application software. Emulation tools and digital analyzers are used during the prototype testing and verification process. An emulator circuit allows the developer to get into the microcontroller while it is plugged into the target system and monitor its performance while making the controller execute a specific part of the low-level application software in a controlled manner. The use of a modern logic analyzer to monitor the performance of the system's emulator gives the designer an outstanding tool for debugging interrupt driven real-time software.

The hardware development steps are summarized as follows:

1. Design the target system hardware: The hardware needed for input and output such as the sensors and actuators is selected. A microcontroller with the required speed and memory resources to handle the I/O and processing requirements is then identified. The microcontroller should perhaps be augmented with extra RAM chips, peripheral controller chips, or I/O expander chips in order to meet the requirements.

2. Simulations of the designed system is then carried out to verify that the performance requirements are being met. Simulators of microcontrollers are usually commercially available in which the microcontroller operations and its I/O interactions are simulated on another computer. Logic circuit simulators can also be used to verify the correctness of the special circuits needed in the design.
3. A hardware prototype is then built: This can take the form of wire wrapped boards to support the required hardware interactions with I/O devices. The prototype can be built in stages in which each stage is verified by the proper testing and verification tools.
4. Low-level software (such as device drivers) is then developed to test the functionality of hardware. This again is done in stages in accordance with the developing hardware prototype mentioned in step 3. This software includes small hardware test programs to verify that the hardware interactions work.
5. Emulators and logic analyzers are then used in the testing and verification process. This includes developing data gathering and triggering circuitry for these tools. A development environment which allows changes to be made in the low-level software or hardware and then tested and verified is very important in reducing the cost and enhancing the quality of the required product. When the developing team verifies the working functionality of the prototype, the application software testing process can then start.

2.2 SIMULATORS

Simulating the instruction execution of microcontrollers can definitely help in the debugging of low-level application software before it can be integrated and tested on the actual hardware prototype. Cybernetic Micro Systems developed simulators for a variety of microcontrollers such as the Sim8096 for the Intel 8096 controller. The DOS compatible simulation environment converts the display in an IBM PC into multi-windows showing the operations in different components of the 8096 as it executes software. The windows consist of the following: a code widow shows the machine code as it is executed; the register window shows the current contents of the various registers and flags which include the major registers, the I/O ports registers, the counters in the programmable timer, and the pointers used to establish what is to be displayed in the memory window; the memory window is used to monitor and modify registers, RAM variables, or port bits during the execution of code; the stack window shows the contents of the stack during execution; the flow window shows the control flow of the program as it executes by displaying a flow chart showing labels and branches; a command window for typing simulator commands; and a help window for on-line help. For examples on the use of such a simulator for debugging a real-time system see the designated reference text for this section.

Hardware circuit simulators can also help in debugging digital and analog circuit designs and to verify timing requirements.

2.3 EMULATION

Simulators, as described in the previous section, are used to simulate the microcontroller operations on a development system such as the IBM PC. An emulator, on the other hand, is used to connect the development system to the target system hardware prototype and carry out monitoring and debugging operations in the actual system environment. An emulator hardware circuit is needed for the following: providing a RAM memory for the microcontroller for application program and data and for data needed by the emulator monitor program; providing reconstructed I/O ports for the microcontroller to substitute for the I/O ports given up by the controller when it is configured with its internal bus lines with the emulator circuit; provide communication between the monitor program and the development system, including the ability to download object coed from the development system to emulator RAM for execution by the controller; permitting single stepping or full speed execution of the application software with capabilities for displaying the effects of instruction executions on the CPU registers as well as setting, displaying, and clearing breakpoint.

Examples of low-cost stand-alone emulators are the Motorola M68HC11EVM evaluation module and the Intel iSBE-96 built to emulate systems based on the M68HC11 or the 8096 microcontrollers, respectively. Such emulators control the target system with a processor of the same type as in the target system. For more information on such emulator see the designated reference for this section pages 436-447, and the manufacturers data sheets.

2.4 LOGIC ANALYZERS

Modern logic analyzers provide the designer with emulation and logic analysis features which give an outstanding view of what the microcontroller is doing while running actual application software. They have the ability to capture data and signals and display them in a multi-window environment. Such devices can be used for hardware debugging, software debugging, hardware/software integration, performance evaluation, system testing, and system optimization. As an example of the power of a modern logic analyzer, consider the case in which an application software runs perfectly by itself, but when it is run in conjunction with the complete instrument or device software, one of its variables becomes corrupted. A logic analyzer can be set to trigger only writes to the variable and to collect not only the data written to that address but also the memory transaction which takes place during the following CPU cycle. This transaction determines the address from which the next instruction is fetched and hence determine the part of the program which caused data corruption.

Earlier emulator/logic analyzer devices, such as the Orion Instruments Universal Development Laboratory, are discussed in the designated reference text for this section pages 454-461.

Modem systems such as the Tektronics GPX logic analyzer offer support for todays microprocessors including the 50 MHz 486, DSPs and high performance RSIC chips. The GPX provides a single-probe-multimeasurment system which allows a variety of measurements without changing probes or double-probing. The low load probing system allows and custom probe adapters enable the monitoring of microprocessor lines both synchronously and asynchronously in the same time. The analyzer triggering step is considerably simplified in the GPX by having 27 pre-programmed trigger set-ups that can be used in the majority of cases and that can easily be customized for specific applications.

The GPX provides the following displays: 4 modes of disassembly (hardware, software, control flow, and subroutine) with high-level language symbols for software debugging; graph mode plots acquired data value vs time which is useful for A/D applications; real-time performance analyzer histograms shows the designer where a program is spending its time; a ROM emulator display allows memory to be examined and changed for quick program patches.

The real-time performance analysis module offers 12 ranges to monitor the microprocessor's bus activities in real-time such that no bus cycle is missed. This is in contrast with traditional statistical performance analyzers which sample the bus periodically hence missing significant amounts of bus activity. Monitoring every bus cycle is crucial when the designer is looking for a random glitch or other single-shot anomaly.

CHAPTER III. REFERENCES

[LAWRENCE&MAUCH 87] Real-Time Microcomputer System Design: An Introduction, McGraw-Hill, 1987.

[HOEDESKI 85] Design of Microprocessor Sensors and Control Systems, by M. F. Hordeski, Prentice-Hall, 1985.

[TOMPKINS&WEBSTER 88] Interfacing Sensors to the IBM PC, Prentice-Hall, 1988.

[Miller 86] Miller, E.M. "RTDs and Thermocouple," PC Tech Journal, 4(6):47, 1986.

[GATES&BECKER 89] Laboratory Automation Using the IBM PC, Prentice-Hall, 1989.

[PEATMAN 88] Design With Microcontrollers, By John B. Peatman, McGraw-Hill, 1988.

[INTEL 93] Embedded Microcontrollers and Processors Vol. II, Intel Literature Sales, 1993.

PART III

REAL-TIME SYSTEMS DESIGN II LABORATORY

REAL-TIME SYSTEMS DESIGN II LABORATORY

I OVERVIEW:

In this laboratory, the concepts learned in the class sequence will be applied to real world situations. Each laboratory will require the development of real-time programs, data acquisition, and hardware control. Also, this lab will assume familiarity with concepts and techniques learned in REAL-TIME SYSTEMS DESIGN I, the first course in this course sequence which dealt with real-time software development. Therefore, CFD/DFDs, structure charts, and Ada Structure Graphs (ASGs) will be used extensively on each project. These charts can be made using the Teamwork Case Tool used in the first course.

All projects will make use of a data acquisition board and the Ada programming environment. It is required that each student have a basic working knowledge of the Ada programming language and a good understanding of microprocessor interfacing.

There will be 5 labs during the semester. Labs 1 and 2 will be done on an individual basis. Labs 3, 4, and 5 will be done in groups of 4. These labs are briefly described as follows:

Lab 1. Getting acquainted with the Ada compiler and supporting libraries.

In this lab, all aspects of the Ada compiler will be explained. A small assignment will be given. This will be an individual assignment and it will have a 1 week duration. At the end of the week, the work will be evaluated by the instructor during the lab session.

Lab 2. Ada programming - tasking, interrupts, low-level I/O, interfacing C and assembly to Ada.

In this lab, advanced Ada programming topics will be covered. These topics will include tasking, hardware and software interrupt handling and generation, low-level I/O, and interfacing C and assembly to Ada. A programming assignment will be given using these concepts. This will be an individual lab and it will have a duration of one week. At the end of the week, the program will be evaluated by the instructor during the lab session.

<u>Lab 3.</u> Sensor Monitoring System.

In this lab, a simple analog monitoring system will be constructed. Then the PC will be used to perform various tasks such as data logging, real-time display of data, and error checking (see lab handout for complete details). This will be a group lab and have a duration of 2 weeks. There will be 4 items due: (1) system design (hardware and software), (2) ASGs and software implementation, (3) hardware design and implementation, and (4) system integration and testing. For this assignment, the system design and software design and coding will be due at the end of the first week. The hardware design and implementation and the system integration will be due at the end of the second week. On the due day, a meeting will be scheduled with the instructor and group. Here the group will present their work to the instructor. At this meeting the instructor will evaluate the status of the group and the quality of work. Also, suggestions on improvements may be given. At the end of the second week the same type of meeting will be scheduled. In this meeting the final system will be presented and the instructor will evaluate it on the basis of prior work, completeness, and usability.

<u>Lab 4.</u> Automated Train System.

The lab will involve the simulation of an automated train. Various hardware components will be used to simulate the operation of the doors, climate control, train in motion, etc. The software will be required to show detailed information about the status of the train and allow for user control over many aspects of the trains operation. (see lab handout for complete details) This will be a group lab and have a duration of 3 weeks. There will be 4 items due: (1) system design (hardware and software), (2) ASGs and software implementation, (3) hardware design and implementation, and (4) system integration (see REQUIRED DOCUMENTATION for details). For this assignment the system design and software design and implementation will be due at the end of the first week. The hardware design and implementation will be due at the end of the second week. Finally, at the end of the third week the complete system must be integrated and tested. At the end of each week, a design review will take place and the items listed above will be due. The instructor will evaluate the project on the basis of prior work, completeness, and usability.

<u>Lab 5.</u> Space Shuttle Simulation.

This is the most involved and final lab. It will entail the development of a space shuttle simulation. Hardware will be used to simulate the motion of the shuttle, thrust, skin temperature, etc. The software will be required to provide detailed information about the shuttle including altitude and trajectory (see lab handout for complete details). This will be a group lab and have a duration of 4 weeks. There will be 4 items due: (1) system design (hardware and software), (2) ASGs and software implementation, (3) hardware design and implementation, and (4) system integration (see REQUIRED DOCUMENTATION for details). For this assignment the system design will be due at the end of the first week. The software design and implementation will be due at the end of the second week. The hardware design and implementation will be due at the end of the third week. Finally, at the end of the forth week the complete system must be integrated and tested. At the end of each week, a design review will take place and the items listed above will be due. The instructor will evaluate the project on the basis of prior work, completeness, and usability.

1.1 DESIGN REVIEWS:

At the end of each week, a meeting will be scheduled with each group and the instructor. In this meeting, the group will present the assigned materials and discuss it with the instructor. The work will be critiqued, and suggestions may be made on how to improve upon it.

1.2 REQUIRED DOCUMENTATION:

For each project, four pieces of documentation will be required. They are: (1) the system description, (2) software design description and implementation, (3) hardware design description and implementation, and (4) the final report.

<u>SYSTEM DESCRIPTION</u>:

The system description will be comprised of a hierarchy of hardware block diagrams detailing every aspect of the system and a complete software description will include

CFD/DFDs, followed by structure charts. At this point the hardware design should be ready to go to schematics and the software design to ASGs.

SOFTWARE DESCRIPTION AND IMPLEMENTATION:

The software description and implementation will be comprised of the ASGs for each module and the corresponding code. At the completion of this step, the software should be ready for integration with the hardware modules.

HARDWARE DESCRIPTION AND IMPLEMENTATION:

The hardware description and implementation will be comprised of all schematics for the system and the corresponding circuitry. At the completion of this step, the hardware should be ready to for integration with the software.

FINAL REPORT:

The final report for each lab will be composed of the following sections:

(1) overview of project

- what was the problem?

(2) system description

- how was the problem solved?
- what hardware blocks were needed? (level 1 block diagram)
- what software blocks were needed? (context diagram and CFD/DFD 0)
- how will hardware and software work together?

(3) detailed software description

- general description of the approach used.
- detailed description of CFDs/DFDs for all levels, and all the structure charts for

the design.

(4) detailed hardware description

- general description of the approach used.
- what type of components were used and why?
- detailed block diagrams for all levels
- how was the actual system simulated by the hardware?

(5) detailed description on how to use of the system

- what buttons due what?
- what does the PC screen tell the user?
- what type of control does the user have?

(6) conclusions

- did it work? why or why not?
- possible improvements.

(7) appendix

- hardware block diagrams
- schematics
- software diagrams
- ASGs
- software source code

1.3 GRADING: -

Each member in the group will receive the same grade. Only if it is noted that an individual is not participating in the projects development will this change, i.e., individual accountability will be enforced. Each project will be graded based upon the work presented in the design reviews, the final report, and the completeness of the project.

At each design review, the progress of the group and the quality of the work will be evaluated. Complete functionality will not be the main component of the grade. A valid attempt and reasons for the path chosen will compose of the bulk of the design review grade.

The final report will be graded upon the adherence to the above criteria and completeness. The report should be written as if the reader knows nothing about the project.

The only assumption about the reader that can be made is that he/she is an engineer and understands the concepts the project employs. A person should be able to read the report and understand how and why it was implemented in the way it was and should be able to run the project without difficulty.

The complete project will be graded on the basis of whether or not it accomplishes the assigned tasks, and if it is friendly to operate. A friendly system is one that requires little to no instruction to operate. A good user interface can make this easy to achieve.

1.4 HARDWARE AND SOFTWARE COMPONENTS AVAILABLE IN THE LAB:

The laboratory used will be equipped with PCs, each containing a data acquisition board which will be used in all projects. All required software will be available in this lab. The software includes TeamWork, Borland C++ 3.1, an Ada compiler, all drivers for the data acquisition board, and an event-driven user interface library.

The data acquisition board will be equipped with a minimum of 8 analog input lines, 8 digital I/O lines, 2 analog outputs, and a timer output. Any combination of these ports may be used.

The drivers for this board will allow direct control of the ports. They can be simply called as subroutines from the program.

The event-driven user interface library will contain all the needed components to create a mouse driven graphical user interface. The library will contain callable subroutines that can produce mouse control, windows, buttons, and input boxes. From these basic components, new elements may be created. Also, the basic framework for an event-driven program will be included in the library. With this the user can have complete control of the environment. Thus eliminating prompts and waiting for the program to ask for input.

Lab 1. ADA COMPILER OVERVIEW

OVERVIEW:

During this lab, two things will be covered. The first is the use of the Meridian OpenAda DOS compiler. The second is a simple programming assignment which will utilize various libraries such as the graphics library, and the DOS environment library. Each group will receive a copy of the manual for the compiler.

This handout contains general information that is needed to get you underway. The lab instructor will go into more details on specific aspects of the compiler.

MERIDIAN OpenAda DOS COMPILER v4.1.4:

During this lab we will not teach you everything about this system. We will show you enough for you to get started. Each group has a copy of the manual; so use it. The manual has complete documentation on all aspects of the system. Also included in the manual are the specifications for all the packages.

OpenAda supports protected mode programming, math coprocessors, and graphics. There is an interactive programming environment called "ACE". Also, there is an interactive debugger called "MADE".

OpenAda is very close to the VAX Ada that you already know. There are some additional libraries that you need to investigate. These packages include: the Ada graphics utility library, math library, and the DOS environment library. These 3 libraries will be used extensively during the semester.

ENVIRONMENT SUMMARY:

The environment is a windowing, mouse driven environment. If you are familiar with Borland's or Microsoft's environments, it will take some time to adjust to this one. ACE is awkward at first, but can be tolerated with time.

To enter the environment type "ACE". Once in, it will ask for a file name. Two things can be done. The first is to enter the name of a file on the drive, or the name of a new file and hit return. The second is to press return, and a file listing of the current directory will given. Once the file is loaded, the left mouse button will activate a floating menu that contains the elements of a standard menu bar. The right mouse button will enter carriage returns.

Things that you may have never seen:

(1) If you entered extra carriage returns, and would like to delete them, the backspace and delete keys will not do it. The only way to eliminate them is to cut them out. To do this, place the cursor on the extra line and select 'cut' from the menu.

(2) You can not drag and highlight a block of text. To do this go to the beginning of the block, activate the menu, select edit, then cut and paste, and finally mark lines. Then move the cursor to the bottom of the block. Now you can cut and copy.

(3) To unblock something, you must go under the cut and paste menu and select unmark.

(4) To manipulate a window, you must use the window selection in the menu. The mouse can not make any alterations on the window.

(5) If you have a line that you wish to break into two lines, placing a carriage return at the point you wish to break the line will not work. To do this, place the cursor at the point of the brake. Go to the Insert and Change menu and select Break Line. This will split the line into two sections.

(6) The ANSI standard for Ada is included in the help files.

ASPECTS OF THE COMPILER:

Before entering the ACE programming environment, a library must be created. To do this use the "newlib" command. From the command line, enter the command and a file named "ada.lib" will be created. As you compile each element of the project it will be placed in the library. Once all elements are compiled, the "MAKE PROGRAM" command may be used. This will compile and link the program. Once all files have been compiled initially, there is no need to recompile them. The "MAKE PROGRAM" command will take care of it for you.

There are many switches that may be added to the compile command. See the manual for a description of them.

ASPECTS OF THE DEBUGGER:

To use the debugger, the '-fD' switch must be used. To do this from the ACE environment go under options, and then compiler options. Turn on the debugging code generation. Then exit ACE and run MADE. The syntax is "MADE [name of exe]". This will take you to the debugger and load the file. You will find the debugger environment friendlier than ACE. See the manual for exact usage of the debugger.

ADDITIONAL PACKAGES SUPPLIED :

In addition to the libraries that came with the compiler, a graphical user interface library, a mouse library, and several procedures to implement event-driven programs are available. The documentation for these libraries are included in the back of the compiler manual to be handed in the lab.

The graphical user interface library contains various forms of windows and buttons. The windows that are available include: output windows, input windows, and windows containing a user defined number of buttons. These windows can be overlaid to create screens containing buttons and input ports that can launch various functions from the push of a button, display data, receive information from the user, and send messages to the user. The windows and buttons are created by simply calling a procedure with the size and position of the window or button. The buttons can be accessed by either a mouse click or keyboard entry. If more detailed windows are desired, new windows can be derived from the standard window and button packages.

The mouse library contains the routines necessary to provide mouse control. With this library, one can find the mouse coordinates, show and hide the mouse, set mouse sensitivity, change the mouse cursor, receive button clicks, and many other functions. The mouse library requires a Mircosoft compatible mouse driver to already be loaded on the system. The event driven procedures are the frame work for an event driven program.

An event driven program is one that is available for user input at anytime. The program does not ask for input from the user (i.e. there is not a single prompt). The program gives the user multiple areas of input on the screen. The information desired in each window can then be entered at anytime. This type of program gives the user more control over when and how a program will operate. There is no need to wait for a prompt. An example of this type of interface is Microsoft Windows.

ASSIGNMENT:

THIS IS AN INDIVIDUAL ASSIGNMENT.

This assignment will allow for the creation of an event-driven program. It will use the event-driven library supplied, along with the DOS and graphical libraries provided in the OpenAda environment. The things done in this assignment will be used again in later labs.

Create an event-driven program using the graphic interface library and mouse library that will read a provided text file containing numbers and plot each number versus time in a strip chart format. When the end of the file is reached, loop around and start over. The screen should provide an entry location for the file name, start, stop, and end buttons, and the graph. At any time the user can change the file name. When start is pressed, the program will switch to the new file and begin to plot the data in the new file. The transition should look as if no file switch ever occurred.

LAB 2. INTERFACING C AND ASSEMBLY TO ADA

OVERVIEW:

The goals of lab2 are:

1. To be familiar with Ada tasking, and low level I/O programming and interrupts;
2. To be able to interface Assembly or C routines to Ada

You have to use pragma **interface** to make calls to subprograms written in 80x86 Assembly language or Microsoft-C from Meridian Ada programs.

See Maridian Ada Compiler User's Guide, Chapter 15 Pragma Interface, for more details about formal description and calling

conventions.

To interface Assembly program (under Turbo Assembler environment) from your Meridian Ada program, you have to follow the following steps:

1. To have object code for your assembly program use:

   (1). TASM YOUR_ASSEMBLY_FILE_NAME.ASM

   (2). TLINK YOUR_ASSEMBLY_FILE 2.

To compile your Meridian Ada program type the following:

   (1). NEWLIB

   (2). ADA YOUR_ADA_FILE_NAME.ADA

3. To link your program use:

   (1). bamp -r -i your_ada_file_name

   (2). link your_ada_file_name your_assembly_file_name

-r -i: The data segment is renamed DATA.

Now you may run your Ada program with interfacing to Assembly program as follows:

YOUR_ADA_FILE_NAME

To interface C program (under Microsoft-C environment) from an Ada program, you have to follow the following steps:

1. To have object code for your C program key in the following:

   cl /AL /Gs /c your_c_file_name.c

   /AL: Compile the C program with MS C using large model code.

   /Gs: Inhibit stack checking, which would drag in MS C run-time libraries.

   /c: Hold off linking.

2. To compile your Ada program type the following:

   (1). NEWLIB

   (2). ADA YOUR_ADA_FILE_NAME.ADA

3. To link your program use:

   (1). bamp -r your_ada_file_name

   (2). link /nod /batch your_ada_file_name

   your_c_file_name

   /nod: Use no default libraries;

   /batch: Ignore missing files.

ASSIGNMENT:

I Interfacing Assembly to Ada:

1. Using the appropriate DOS/BIOS functions, write an Intel Assembly program to implement an interrupt service routine that will get the system time, maintain your own clock, and display your time on the screen.
2. The output of your program should be as follows:

   Hours : Minutes : Seconds
3. The interrupt servicing routine consists of four main parts:
   a. Using DOS function 21h, function 2ch to get the system time.

   Hours in ch, Minutes in cl, Seconds in dh. You have to save them in predefined variables.

   b. To calculate, and maintain your time in Hours, Minutes, and Seconds in binary code.

You may use the following algorithm to calculate, and maintain your own time from an interrupt of an average 18.2 times per second:

```
Initialization of Count( to 18, 19 or your choice)
Decrease Count
IF     Count  0
THEN return from interrupt
ELSE Increase Seconds
IF     Seconds = 60
```

```
THEN Increase Minutes
Reset Seconds to 0
IF      Minutes = 60
THEN  Increase Hours
Reset Minutes to 0
IF      Hours = 13
THEN Hours = Hours - 12
END_IF
END_IF
Set count to 18 ( 19 or your choice)
Calculate ASCII characters for time string
Display  your time
END_IF
```

c. To calculate the time ASCII character equivalent to display the time. You may use the following algorithm to get the ASCII character equivalent:

```
mov  ax, 00 -- clear ax register for dividing
mov   cl, 10 -- place decimal 10 in cl
mov   al, [Hours] -- Place binary number Hours in al
div   cl -- divide by 10 to get BCD
add   ax, 3030h -- convert to ASCII
```

d. Using DOS interrupt function 21h, function 9 to display your time on the screen.

4. Write your own Meridian Ada program which will use Pragma **interface(assembly, subprogram_name,"link_name")** to interface your Assembly unit. See the example in page 108 of "Meridian Ada Compiler Guide".

5. Compile, link, and run your program.

II Interfacing C to Ada

1. Write your C program unit to fulfil the same objective as you did in the Assembly program.

2. Write your own Ada program which will use Pragma **interface(microsoft_c, subprogram_name,"link_name")** to interface your C unit. See the example from page 110 of "Meridian Ada Compiler Guide".

3. Compile, link, and run your program.

LAB 3. SMART SENSOR CONTROL SYSTEM

OVERVIEW:

This project will involve the development of the hardware and software for a sensor network. This project will be implemented for an Intel 80x86 based PC.

The system to be designed is a two sensor network that will have the following capabilities: (1) to detect changes in air temperature, (2) give warnings when the temperature reaches certain levels, (3) store all data on the PC, (4) give visual indication of the highest value, (5) indicate which sensor has the highest value, (6) strip charting of the data, (7) send highest value to a remote terminal via RS-232, and (8) check for proper operation of the system.

The system will have two parts. The first is a stand-alone analog system that will signal when the temperature reaches the warning or critical levels. The second part is the PC. The PC is used for the following: (1) verify the operation of the analog circuits, (2) display which sensor has the highest reading (via hardware), (3) display the highest temperature, (4) store all values from both channels, (5) strip chart both channels, and (6) send the highest value to a remote terminal via RS-232. The PC could be used to do both parts easily, but, in the real world, there are federal regulations that do not allow the use of microprocessors as the main source of detection in some situations (e.g., inside of coal mines).

HARDWARE: (see block diagrams and schematics at the end of the lab description)

ANALOG:

The analog circuitry needed is minimal. Each sensor must be connected to an amplifier circuit. This is done to boost the millivolt reading up to a level that makes comparisons easier. Next the maximum value from the two amplifiers must be determined. The is done simply be using two transistors with the base of each tied to a different amplifier and the emitters tied together. This value is then sent to two comparitors. Comparitor #1 will check for the warning temperature; while comparitor #2 will check for the critical temperature. The output of each comparitor will be tied to an LED to indicate that the value has been obtained. This circuit should be capable of operating **without** the PC.

PC INTERFACE:

The first piece of hardware needed is the A/D convertor. For this project, 7 A/D channels will be used. The A/D will be connected to the follow points in the analog circuit: (1) sensor #1 output, (2) sensor #2 output, (3) output of amplifier #1, (4) output of amplifier #2, (5) determined maximum level, (6) output of warning comparitor, and (7) output of critical comparitor.

From the placement of the connection points it can be determined if any component in the system has failed. Also, 3 LEDs must be interfaced to the PC. LEDs 1 and 2 will indicate which sensor has the highest value. The third LED will indicate when a problem has been discovered. The malfunction LED should be distinguishable from the other LEDs by blinking it when a malfunction is detected.

SOFTWARE:

The software for this project has 5 sections: (1) data acquisition, (2) interface screen, (3) data storage, (4) diagnostics, (5) LED indicators, and (6) serial communications.

DATA ACQUISITION:

Data should be read from the sensors continuously. As a new data point is read, it's value and the sensor number should be place in a location available to any other routine that my need it.

INTERFACE SCREEN:

The PC screen will be divided into 3 sections. The first will be a message area. In this area all information sent be to the user will be displayed.

The messages will consist of explanations of any error conditions found, and the state of the sensor reading (i.e. normal, warning, critical).

The second section of the screen will be a strip chart running in real time. This chart will be continuously updated with each new data point. Both sensors will be graphed on the same axis, with each sensor distinguished by a different color. For this project the axis will be temperature vs. time. This chart should be easily read, and contain at least 320 data points. With this many data points and running in VGA 640x480 mode, the chart will be as wide as half the screen.

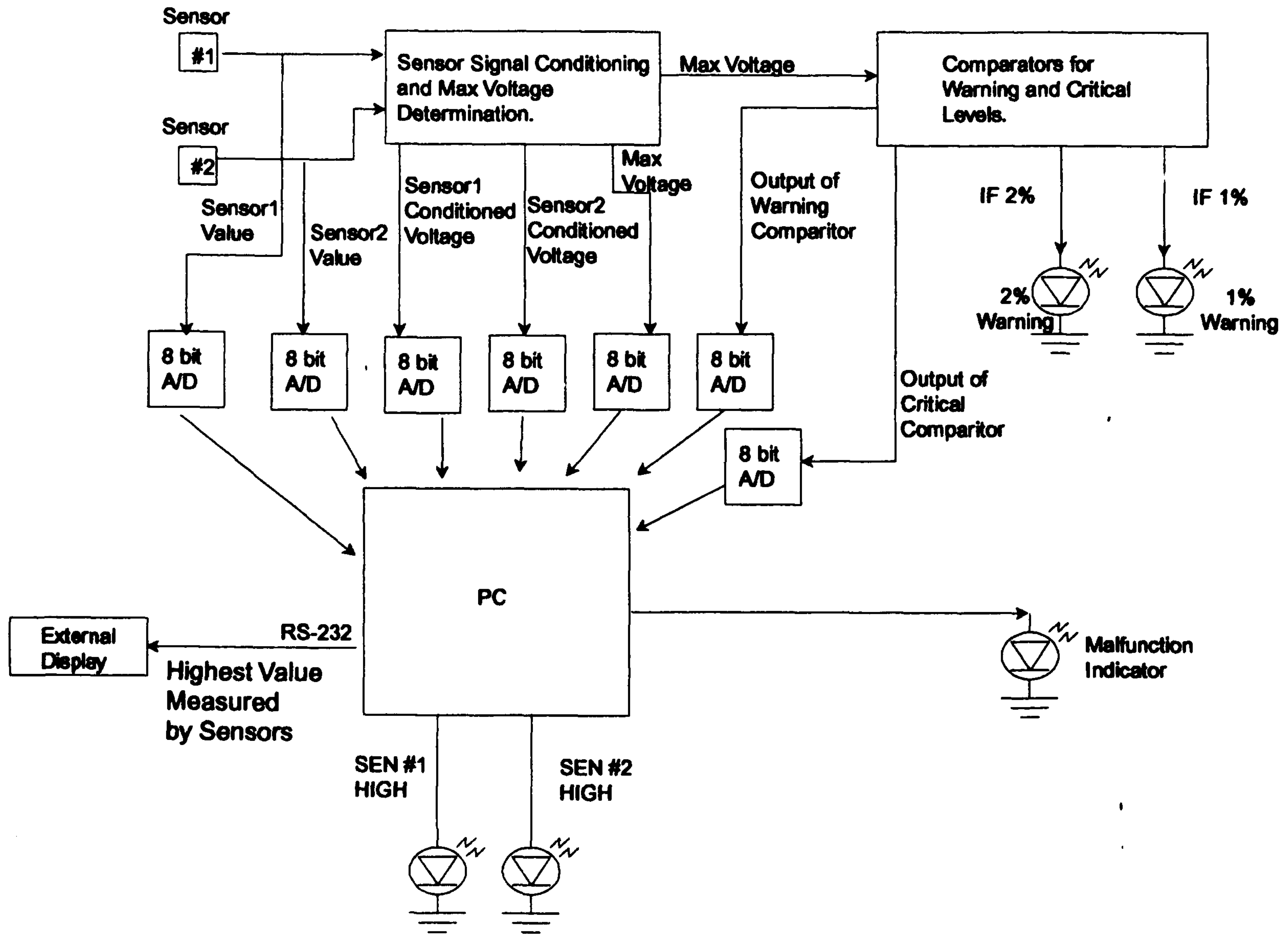
The third section of the screen will display the highest temperature read. The temperature should be accurate within 1/10 of a degree and be updated constantly.

DATA STORAGE:

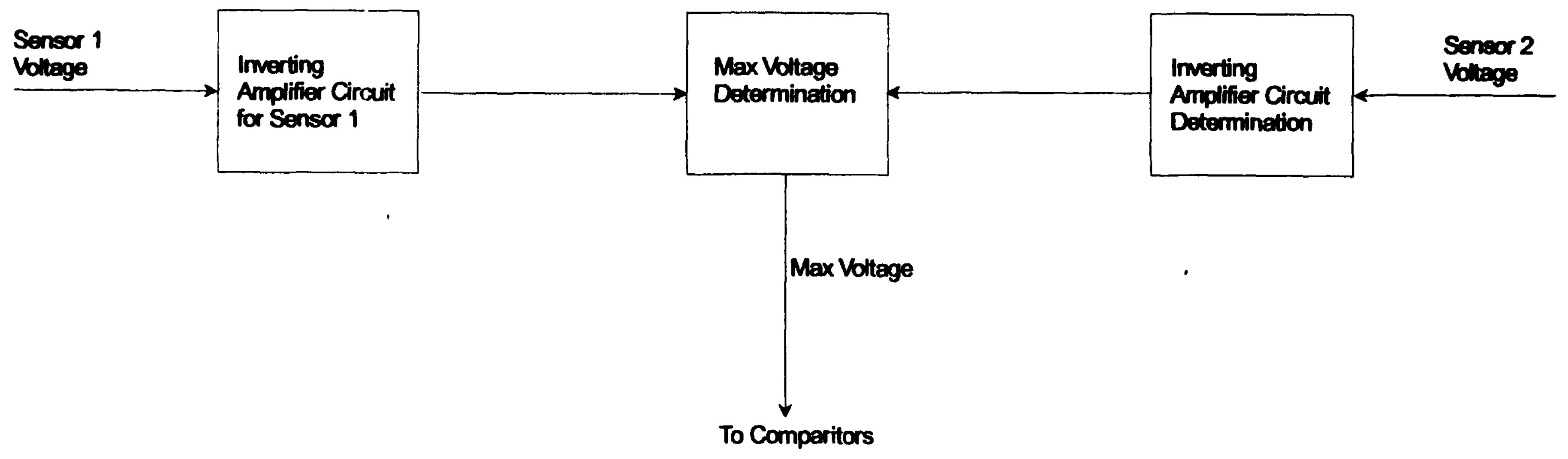
The values from both sensors will be stored in a file on the PC's harddrive. This file should contain the sensor number and the value from that sensor. Data should be stored approximately every 1 second.

DIAGNOSTICS:

The information for the diagnostics comes from the points throughout the analog circuit that the A/Ds are connected. From comparing this data, each component in the circuit can be judged. When comparing the actual sensor values with those values that follow the amplifiers, the amount of amplification must be subtracted or added to the respective signal. If a problem is discovered, a message warning the user of the problem and an explanation of where the problem is and how to fix it must be displayed. Also, the LED signaling a malfunction must be activated (by blinking it). This LED will remain active

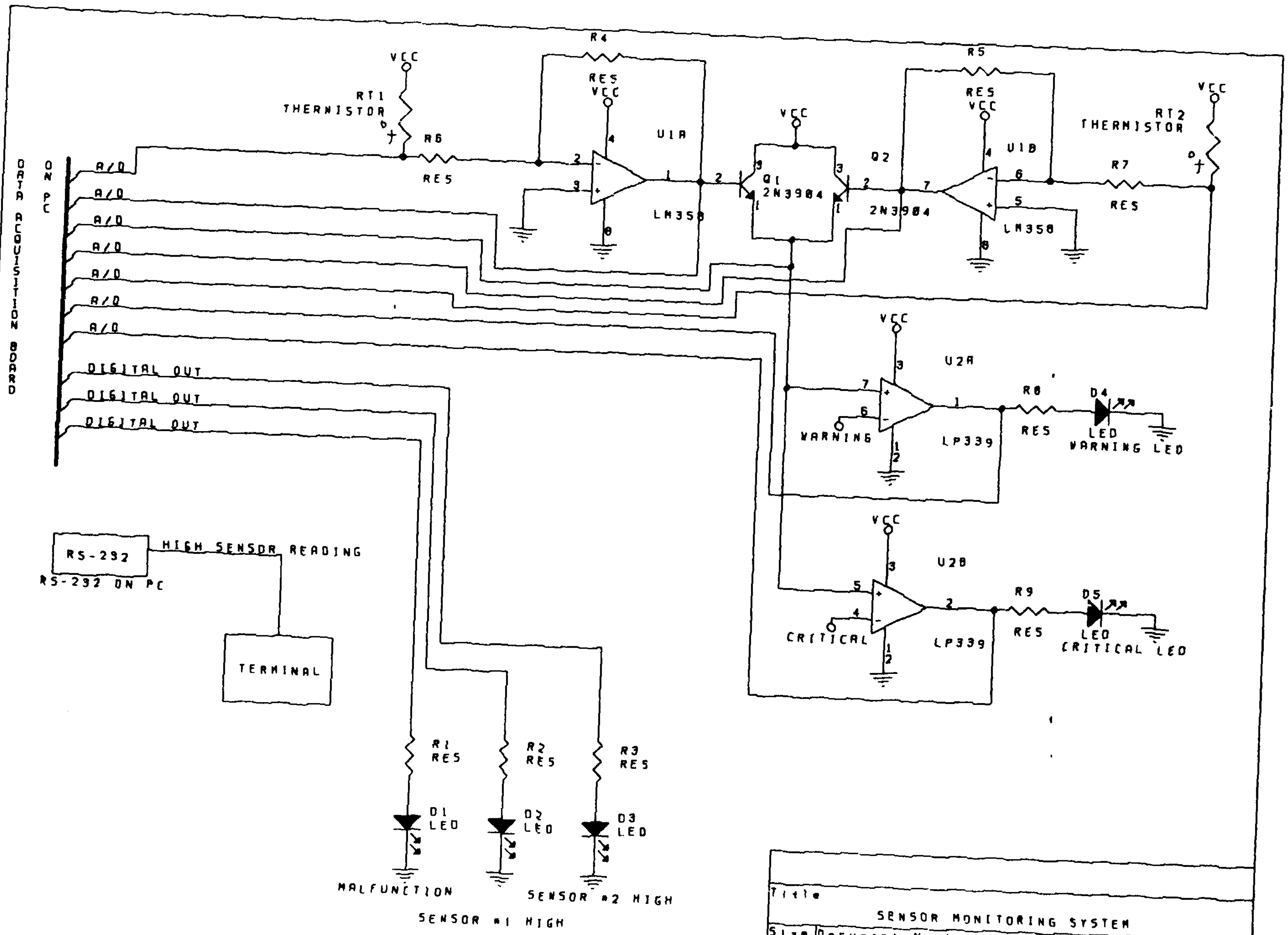

Sensor
#1
Sensor
#2
Sensor Signal Conditioning and Max Voltage Determination.
Max Voltage
Comparators for Warning and Critical Levels.
Sensor1 Value
Sensor2 Value
Sensor1 Conditioned Voltage
Sensor2 Conditioned Voltage
Max Voltage
Output of Warning Comparitor
IF 2%
IF 1%
2% Warning
1% Warning
Output of Critical Comparitor
8 bit A/D
PC
RS-232
External Display
Highest Value Measured by Sensors
Malfunction Indicator
SEN #1 HIGH
SEN #2 HIGH

SENSOR MONITORING SYSTEM
BLOCK DIAGRAM: SYSTEM DESCRIPTION.


Sensor 1 Voltage
Inverting Amplifier Circuit for Sensor 1
Max Voltage Determination
Inverting Amplifier Circuit Determination
Sensor 2 Voltage
Max Voltage
To Comparitors

SENSOR MONITORING SYSTEM

BLOCK DIAGRAM: SENSOR SIGNAL CONDITIONING AND MAX VOLTAGE DETERMINATION.



Title	SENSOR MONITORING SYSTEM	
Size A	Document Number	REV
Date:	August 5, 1993	Sheet 1 of 1

until the problem is corrected.

LED INDICATORS:

The 2 LEDs that indicate which sensor has the high value and the Malfunction LED. The high value LEDs will be activated when the sensor corresponding to the specific LED has the highest value. These values are to be continuously updated. The Malfunction LED will be activated when a malfunction is discovered in the system.

SERIAL COMMUNICATION:

The highest value will be sent to a remote terminal via the PC's onboard serial port. The information sent is to be continuously updated.

This lab will be a group effort lab and have a duration of **two WEEKS**.

DUE DATES:

Due at the end of the **first week**:

(1) system design (hardware and software), and

(2) DFDs/CFDs, ASGs, and software implementation

Due at the end of the **second week**:

(3) hardware design and implementation **AND**

(4) system integration and final report

LAB 4. AUTOMATED TRAIN CONTROL SYSTEM

OVERVIEW:

This project will consist of the development of software to control the instruments of an automated train system, and various hardware components that will work interactively with the software to simulate an automated train.

This simulation will allow for the control and monitoring of various subsystems of an automated train. These systems include: choice of operating modes, climate control, opening and closing of doors, operation of the motor, distance measurements, and error checking. The user will have an interface screen showing the status of the train and allowing for the input of various information needed during the run (see **SOFTWARE** -- INTERFACE SCREEN or **SOFTWARE** -- TRAIN OPERATION for further information).

Before the train can operate, a track must be constructed. This will be done by having the user to provide the number of stations, the distance between stations, and the distance required for the train to stop at each station. Once the track has been constructed, the simulation will begin. The train will be able to operate in two different modes. Mode one will be a continuous travel mode. While operating in this mode, the train will travel to all stations on the track and perform the exiting and boarding of passengers. Mode two will be a direct travel mode. In this mode, the train will continuously travel around the track without stopping until a request has been made by a "passenger". The request can come from two places: (1) a passenger currently on the train, or (2) a passenger waiting at a station.

The request from a currently boarded passenger will be received through a button interfaced to the computer (see **HARDWARE** section for further explanation). When this signal has been received, the train should stop at the next station. The request from a passenger waiting at a station will be received from the keyboard. The user will enter the name of the station to be stopped at next. The train can only stop at the requested station if the distance to the station is sufficiently long enough to stop.

The train will have five major hardware components: (1) the next stop button, (2) a temperature sensor for the climate control, (3) the opening and closing of the doors, (4) motor operation and distance measurements, and (5) error checking of items 2-4 (detailed explanations can be found in the **HARDWARE** section). The next stop button, as explained above, will allow for a "passenger" to request a stop at the next station. The temperature sensor will continuously measure the air temperature, when the temperature goes above or below a set value, the appropriate action should be taken, and an indication to the interface screen of the situation and the action taken. The opening and closing of the doors consist of an LED indicating if the door has been opened or closed. Also, the door subsystem will allow for the checking of a door blockage at the time of closing or a door that has been opened during travel.

The motor and distance measurement will be implemented by the use of a motor and a 555 timer chip. When the train is moving, the motor and the 555 will be activated simultaneously. Each pulse from the 555 will indicate a unit of distance travelled. The error checking will consist of verifying that information is continuously being received from the 555 and temperature sensor, and that the door is not blocked or opened during travel. If any of the errors occurs, the appropriate action will be taken and an indication to the user on the interface screen will be displayed explaining the situation and the action taken.

HARDWARE: (see block diagrams and schematics at the end of the lab description)

The functions simulated will be: the train in motion, distance travelled, door opening/closing, climate control, and "passenger" stop request. This will be done by interfacing various components to the computer, either directly to the bus, or through a data acquisition board.

TRAIN MOTION:

The motion of the train will be simulated through the use of a motor. The motor will be connected to the computer via a D/A converter. When the train is in motion, the motor will be running, if the train has stopped, so will the motor.

DISTANCE TRAVELLED:

The distance traveled will be calculated by activating a 555 timer chip. With each pulse from the 555, the train will have moved a unit of distance. The 555 will be activated when the train is in motion. The output of the 555 will be tied to a interrupt line of the computer. In doing this, special consideration must be made to keep the 555 output below 5V. A switch will be placed in the output line to simulate a loss of the motor.

DOOR:

The door will be simulated by simply lighting an LED connected to the PC. To simulate a door blocked door, or a door opening a button will be tied to the PC. A pressed button will indicate an error, and the state of the train will determine which error. If the train is in motion, the door has been opened. If the train is stopped, the door is blocked.

CLIMATE CONTROL:

The climate control will be a thermistor tied to an A/D convertor and then into the computer. A switch will be placed in the line to simulate the loss of climate control.

PASSENGER STOP REQUEST:

The "passenger" stop will be a button tied to an interrupt control line. When this button is pressed, the train will try to stop at the next station. Before attempting a stop, the distance to the station and the distance to stop must be compared to verify that a stop is possible.

SOFTWARE:

The software required for the system will have four parts: (1) the user interface, (2) the hardware control, (3) train operation, and (4) error checking of the hardware. The user interface will give the user the ability to control certain aspects of the train. The train operation will take all the information from the user interface and "drive" the train. The hardware control will be the routines that read and write to any hardware devices interfaced to the computer. Finally, the error checking will verify that the correct information in being received from each device. All software is to be implemented in Ada and should take advantage of multi-tasking and interrupt handling whenever possible.

USER INTERFACE:

The user interface will have two functions. The first function will be the configuration of the system (i.e. track description) and the second will consist of all aspects of running the train.

The configuration screen must allow for the entry of the number of stations, the distance between stations, and the distance required for the train to stop at each station. These are the minimum requirements, additional information may be added at the discretion of the designer. Once all the information has been entered, the track should be "constructed" and the system should been prepared for activation.

Following the configuration screen will be the control screen. From this screen, the operator will be able to communicate with the train while in operation. The control screen will consist of at least 3 sections: (1) input section, (2) status indicators, and (3) a message window.

INPUT SECTION:

The input section will contain control parameters that can be changed during operation. The minimum requirements are TRAVEL TYPE, STOP REQUEST, and TEMPERATURE SETTING.

TRAVEL TYPE will have two choices: continuous travel, or direct travel (for a description of continuous or direct travel see TRAIN OPERATION). STOP REQUEST will be the means by which a station can be specified as a stop while in direct travel mode. Once a station has been entered, it will be placed on a list of stops in the order of occurrence on the track with respect to the current position. The list of stops will be displayed on the screen. Also, the user must have the ability to add new stops to the list before any previously entered stops have occurred. TEMPERATURE SETTING will control the temperature at which the "climate control" will be activated. This will be the temperature that will be compared with the reading returned by the temperature sensor. The user must have the ability to modify the values of the controls at ANY time.

STATUS INDICATORS:

The status indicator section of the user screen will contain indicators for the door status and the climate status. The door indicator needs to show if the door is currently open or closed. The climate indicator needs to show if the climate control unit is active or not.

MESSAGE WINDOWS:

The message window will be where the user will receive messages from the system. The window should report errors, the next destination, and the current status of the train, travelling or stopped.

HARDWARE CONTROL:

This set of functions will control the hardware interfaced to the computer. The required hardware will be a "next stop" button, a temperature sensor, a door simulation circuit, a motor, and distance measurement circuit.

The software for the "next stop" button will consist of a means for determining if the button has been pushe . If the button was pushed, a message needs to be sent to the TRAIN OPERATION routines to indicate the occurrence.

The temperature sensor will be read at regular intervals. Each reading needs to be passed to the TRAIN OPERATION routines.

The door simulation circuit software must have the ability to open and close the door. This will consist of turning on and off an LED (assuming the circuit described in the **HARDWARE** section is used).

The motor software will access a D/A converter to run the motor while the train is in motion. To meet the minimum requirements, only a single speed will be needed.

The distance measurement software will read a counter mounted on the motor. This counter will send a single pulse for each unit of distance traveled. This distance must be provided to the TRAIN OPERATION routines.

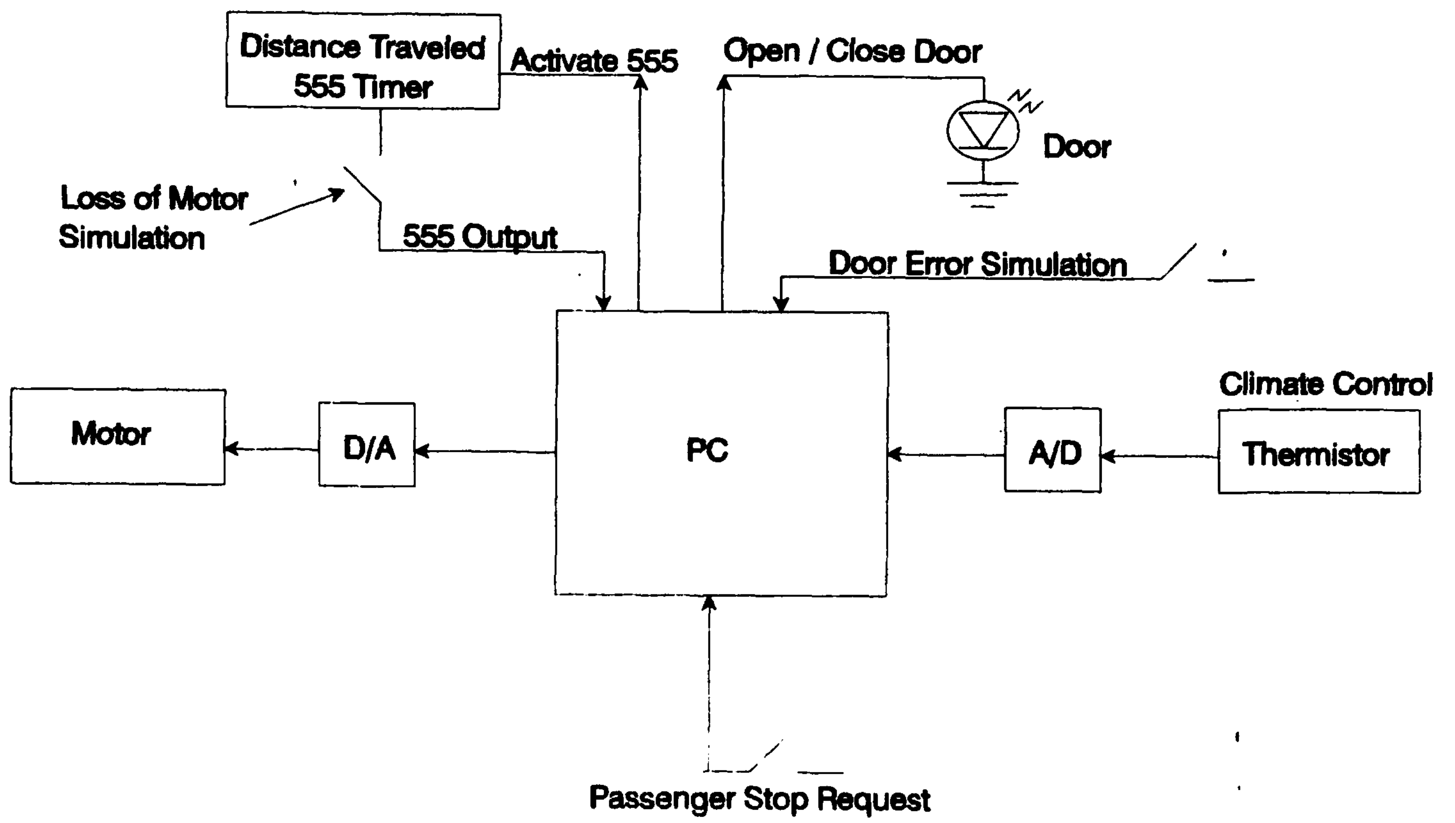
TRAIN OPERATION:

This section will use the information entered by the user and any information gained from the hardware to determine what state the train needs to be in. This set of functions must have the ability to recognize the need to start or stop the train, open or close the door, and turn on or off the climate control. Also, these functions must determine the next stop, distance to the next stop, if the train has the ability to stop at the next station, and if a "passenger" stop request can be fulfilled. Once the information has been evaluated, the functions from the HARDWARE CONTROL will be used to modify the trains status.

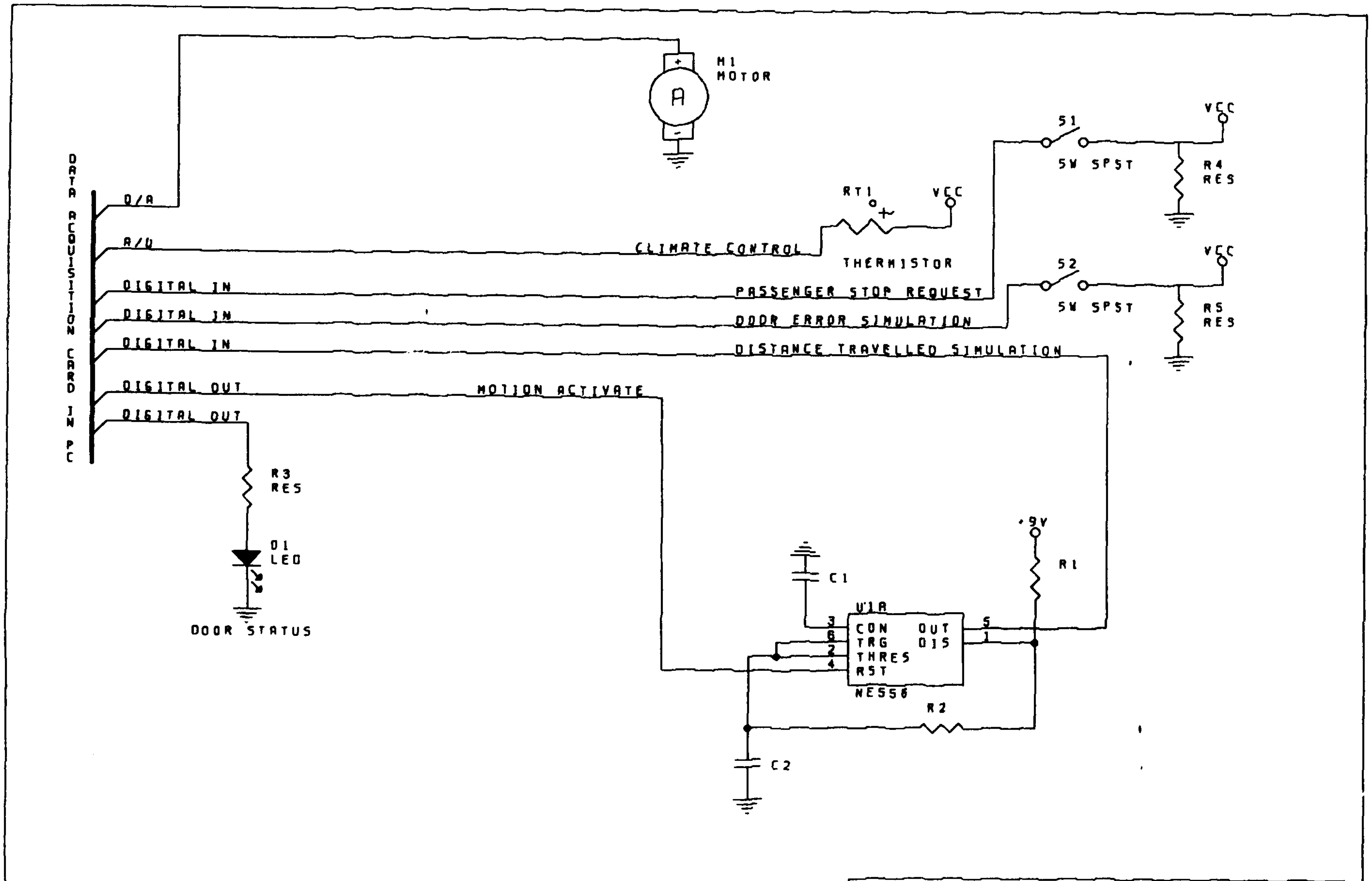
ERROR CHECKING:

The error checking will consist of verifying that data is continuously being received from the motor and temperature sensor, and the door is not blocked or opened.

When the motor is in operation, the software must verify that data is being receive from the distance measurement circuit. If this data is absent, we will assume that the motor has a problem. If a motor problem is discovered, the train should be stopped until the problem is fixed. Therefore, the error checking must continue after a problem has been discovered. An error in the temperature will be indicated by a loss of data. In the case of a temperature error, the "climate control" should be stopped. A door error is indicated when a signal from the hardware indicates that something is blocking the door, if the door is open, or that the door has been opened while the train is moving. If this error occurs while the trained is stopped, the train can not begin until the door closes. If the train is moving and the error occurs, the train is to be stopped immediately and not to resume motion until the door is closed. All error conditions should be reported to the user.



AUTOMATED TRAIN SIMULATION
BLOCK DIAGRAM: SYSTEM DESCRIPTION



Title		
AUTOMATED TRAIN SIMULATION		
Size	Document Number	REV
A		

This will be a group lab and have a duration of **3 WEEKS**.

DUE DATES:

Due at the end of the **first week**:

(1) system design (hardware and software) **AND**

(2) ASGs and software implementation

Due at the end of the **second week**:

(3) hardware design and implementation

Due at the end of the **third week**:

(4) system integration and final report

LAB 5. SPACE SHUTTLE CONTROL SYSTEM

OVERVIEW:

This project will involve the development of software for a space shuttle control system, and various hardware components that will work interactively with the software to simulate physical aspects of the space shuttle.

This simulation will allow for the control and monitoring of various subsystems in the space shuttle. These systems include: pitch and roll (up, down, left, right), velocity, skin temperature, and altitude and trajectory calculations. The user will have an interface screen showing the status of the shuttle. All inputs to the system will come from hardware modules, except the "begin" and "end" commands.

The simulation will allow for control of the 4 directions (left, right, up, down) and the velocity of the ship. The pitch and roll controls will consist of 4 buttons interfaced to the

PC. From reading these buttons, the shuttle can be moved. The shuttle will be represented by mounting a model on a structure containing 2 motors. M1 (motor 1) will be connected to the ship and control the left and right roll. M2 (motor 2) will control the pitch by moving M1 and the ship. The velocity will be controlled by turning a potentiometer tied to the PC via an A/D convertor. The skin temperature (ST) of the shuttle will also be monitored. This value will be a combination of the air temperature, read through a thermistor, and an constant determined by the instantaneous velocity of the ship. If the ST gets to great a warning is to be issued. If it reaches the critical point the ship is "destroyed", and the simulation is over. The altitude and trajectory is calculated by tracking all previous angles and velocities in the vertical plane. The ship will begin at 0 altitude and must "take off". Once underway, if the altitude falls below 0, the ship has crashed and the simulation is over.

The designer must define "actual" values that correspond to the values reported by the hardware. For the direction control, a means of determining the distance turned with respect to the amount of time the button was held. Also, the voltage across the potentiometer must be converted into a velocity. A temperature constant must also be calculated that is related to the velocity of the ship. All "actual" values must be unique for a specific input. (Determination of values is discussed later.)

HARDWARE: (see block diagrams and schematics at the end of the lab description)

The functions simulated will be: (1) flight of shuttle, (2) pitch and roll, (3) velocity, and (4) skin temperature. This will be done by interfacing various components to the computer, either directly to the bus, or through a data acquisition board.

FLIGHT OF THE SHUTTLE:

The ship will be represented by mounting a model on a structure containing 2 motors. M1 (motor 1) will be connected to the ship and control roll of the shuttle. M2 (motor 2) will control the pitch by moving M1 and the ship.

These motors will be tied to the PC through a data acquisition board or by directly interfacing the needed components to the bus.

MOTOR EVALUATIONS:

Data needs to be collected for each motor to determine the angle the model will go through in a set amount of time. From this data and the length of time moved in a specific direction, the degrees per delta time can be calculated with some accuracy. By placing these values into the code, the altitude and trajectory can be calculated for the model. (See **SOFTWARE:**SHUTTLE OPERATION for more information on altitude and trajectory calculations.)

The speed at which the motor is to operate must also be determined. This value is up to the designer, and will be the value used in the motor control routine in the hardware control software.

PITCH AND ROLL:

The shuttle will be controlled by four buttons (up, down, left, right). These buttons will be tied to the PC through a data acquisition board or by directly interfacing the necessary components to the bus.

There is no need for an A/D convertor. The buttons can be constructed so that they provide standard logic levels.

VELOCITY:

The velocity input is simply a potentiometer tied to an A/D convertor. The voltage across the potentiometer will be logged by the PC and the software will match the voltage to a predetermined velocity.

SKIN TEMPERATURE:

The skin temperature is calculated in two parts. The first part is from the hardware, while the second part is determined by the software. The hardware portion of ST is the air temperature. This value is obtained through the use of a thermistor interfaced to the PC. An A/D must be used for interfacing.

SOFTWARE:

The software required for the system will have four parts: (1) the user interface, (2) the hardware control, (3) shuttle operation, and (4) error checking of the hardware. The user interface will keep the user informed of the status of the system and provide begin and exit options.

Begin and exit will be the only simulation inputs received through the software. The shuttle operation routines will collect and send data to and from the hardware through the hardware control routine, track the altitude and trajectory, calculate the skin temperature, and determine the velocity. The hardware control routines will be the means by which a software connection to the hardware will is made. All data sent to or received from the hardware will travel through this set of routines. The error checking must be capable of detecting a loss of thrust and malfunctioning skin temperature sensor. All software is to be implemented in Ada and should take advantage of multi-tasking and interrupt handling whenever possible.

USER INTERFACE:

The user interface will be used to display information to the user. No input other than "begin" and "exit" will be used in the simulation. The user should be kept informed of the follow items: (1) Altitude and angle of trajectory, (2) skin temperature, (3) percent thrust, (4) the time to hit the ground at the current velocity and trajectory, and (5) any other messages deemed necessary.

HARDWARE CONTROL:

The software must be able to receive inputs from the four direction buttons, a potentiometer, and a thermistor. Control of the buttons consist of simply reading when a button is pressed and transferring this information to the SHUTTLE OPERATION. The potentiometer and the thermistor must be tied thought an A/D convertor. Once the data has been logged, it must be forwarded to the SHUTTLE OPERATION routines for processing.

The software must also have the ability to drive both motors controlling the shuttle model. These connections will be made via a D/A convertor. The amount of time to run each motor will be determined by the SHUTTLE OPERATION routines.

SHUTTLE OPERATION:

These routines will use the HARDWARE CONTROL routines to collect all data, send any control signals to and receive any data signal from the hardware. The data collected will consist of: (1) up, down, left, right button presses, (2) voltage on velocity potentiometer, and (3) the skin temperature. From this data, the shuttle must be moved, the altitude and trajectory calculated, the time to ground impact calculated, the skin temperature calculated, and the all information on the user screen must be updated.

BUTTON USAGE (UP, DOWN, LEFT, RIGHT):

The buttons will be used to move the shuttle model. When a button is pressed, the corresponding motor must be activated for the duration of the button press. The distance the model can move in a specific direction will have a maximum angle. For a roll, the value will be approximately ±90 degrees and for the climb approximately ±60 degrees. The method for calculating the angles is described in the description of the altitude and trajectory. These stops must be checked by the software. When an attempt is made to go beyond the values, the system must intervene and not allow the model to move any further. The user needs to be informed of the system override. (NOTE: make sure the motor configuration has enough degrees if freedom to satisfy the above requirements.)

VELOCITY:

Before a velocity can be determined, a velocity per millivolt must be determined. This value is up to the designer. The velocity data is gathered by accessing the potentiometer through the hardware control routines. Once this data has been retrieved, it is a trivial task to find the actual "velocity" of the ship.

SKIN TEMPERATURE:

Before the skin temperature can be calculated, a degrees per velocity unit must be determined. With this value and the instantaneous velocity, a constant can be calculated that represents the additional heat from air friction. To get the other component of the skin temperature, the thermistor value needs to be gotten through the HARDWARE CONTROL routines. Once both values have been acquired, the skin temperature is simply the sum of two values. Two values must be chosen to represent the warning temperature (WT) and the destruction temperature (DT). When WT has been reached, a message should be displayed to the user. When the skin temperature reaches DT, the shuttle is "destroyed" and the simulated ended.

ALTITUDE AND TRAJECTORY:

The altitude and trajectory will be calculated by this set of routines. The trajectory will be the current vertical angle. It is determined by keeping a running total of the amount of time the up and down buttons have been pressed. The time corresponding to the up button being pressed is positive and time for the down button is negative. With this time and the degrees per delta time for the vertical plane, the trajectory is easily calculated.

The altitude will be calculated using the instantaneous trajectory and velocity as defined by the equation $Y_i=V_i*\sin\Theta_i$ (i=instantaneous values). A running total must be kept to maintain the correct value. The altitude will be calculated continuously. To simplify this task, the function should be implemented as a timer interrupt function. The altitude must also be monitored for a zero value. If this occurs, the shuttle will crash and the simulation will end. When the altitude begins to approach zero, the computer is to take control and attempt to level the ship before it impacts with the ground. Since the motor can only move so fast, the success of the rescue attempt will depend on the velocity, which the computer has no control over. This means that the computer can change the trajectory of the ship only as fast as the motor can move the model.

ERROR CHECKING:

The error checking will detect a loss of thrust and malfunctioning skin temperature sensor. The loss of thrust will be signified by receiving a zero potentiometer voltage while at an altitude greater than zero. A message should be conveyed to the user. A malfunctioning skin temperature sensor will consist of a loss of the reading from the thermistor. Once again, the appropriate message should be displayed.

This will be a group lab and have a duration of **4 WEEKS**.

DUE DATES:

Due at the end of the **first week**:

(1) system analysis and design (both hardware and software)

Due at the end of the **second week**:

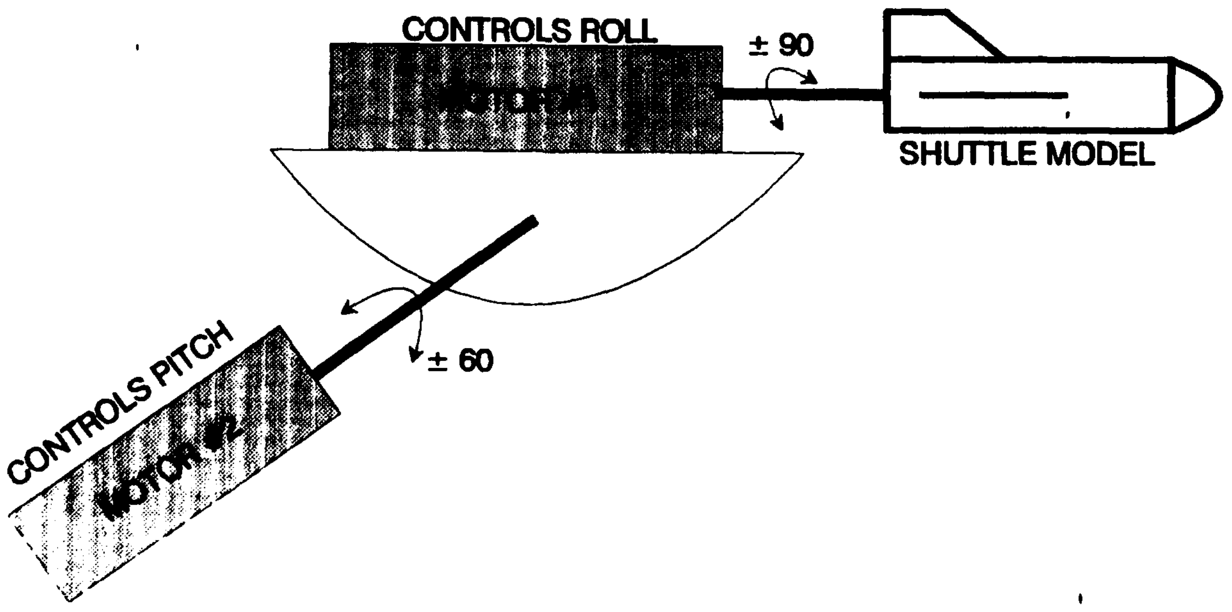
(2) ASGs and software implementation

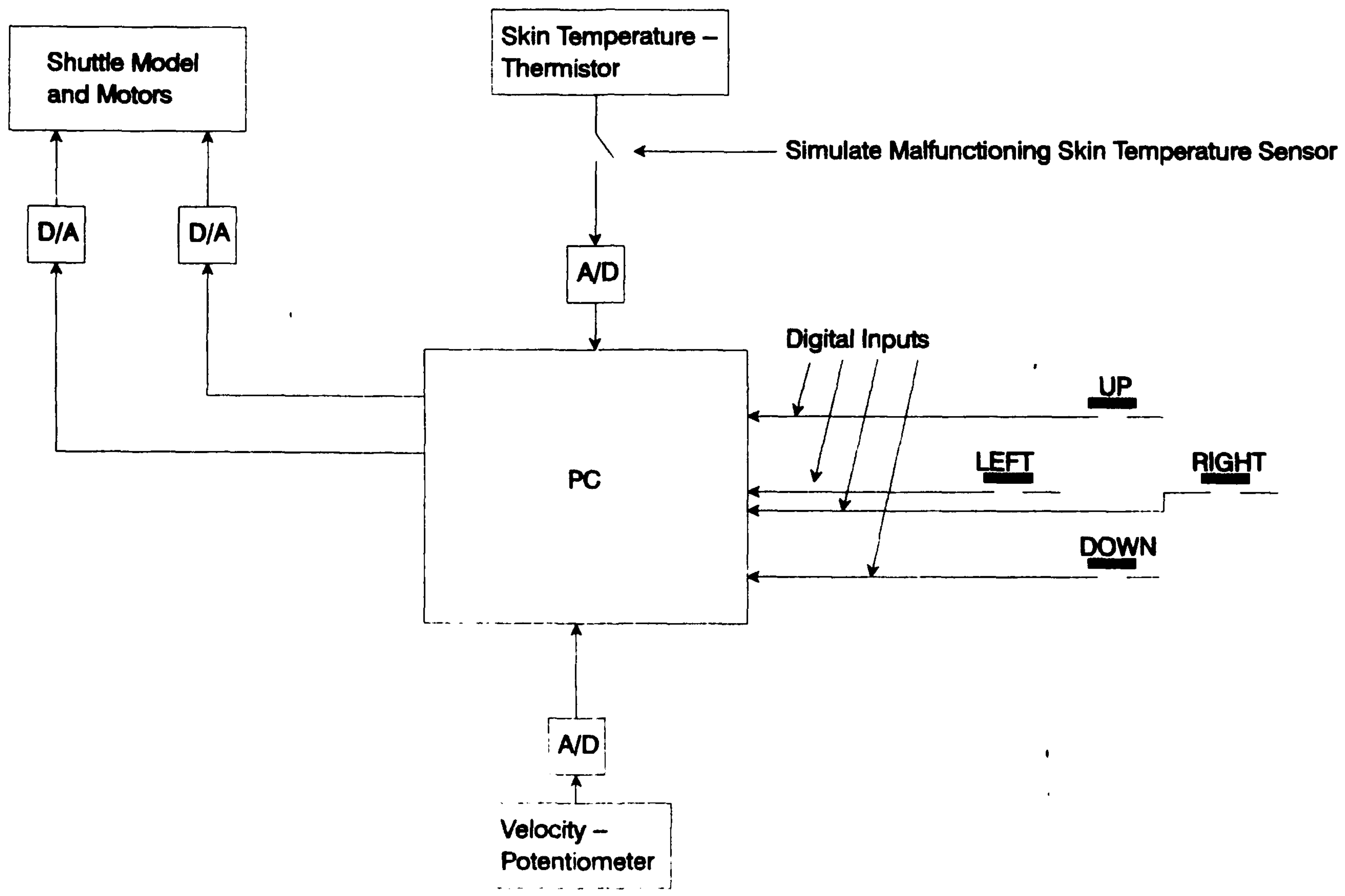
Due at the end of the **third week**:

(3) hardware design and implementation

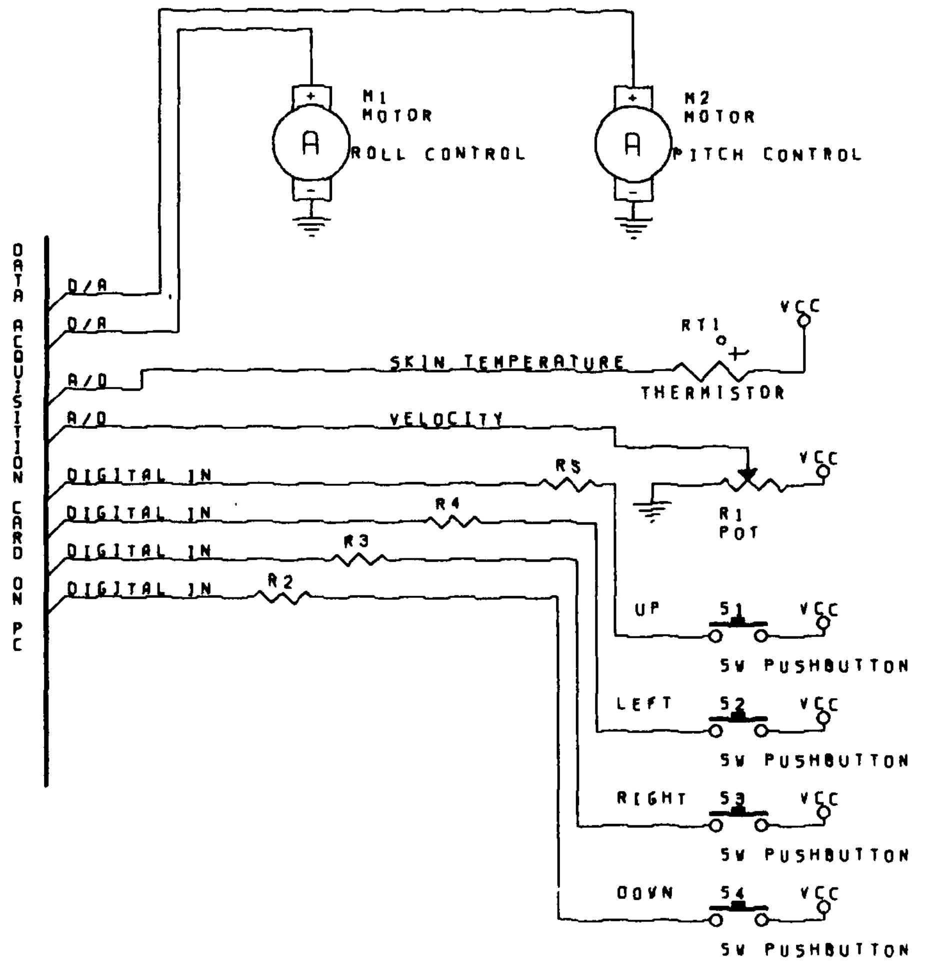
Due at the end of the **fourth week**:

(4) system integration, testing, and final report.


CONTROLS ROLL
MOTOR #1
± 90
SHUTTLE MODEL
± 60
CONTROLS PITCH
MOTOR #2



SPACE SHUTTLE SIMULATION



Title
SPACE SHUTTLE SIMULATION

PART IV

APPENDIX

APPENDIX

Intel EMBEDDED MICROCONTROLLERS AND PROCESSORS

1. MCS-48 FAMILY

The Intel MCS-48 family consists of the 8748H and 8749H EPROM; 8048AH/8049AH/8050AH ROM; and 8035AHL/8039AHL/8040-AHL CPU only single component microcomputers.

1.1 MCS-48 Single Component System

Taking 8048AH EPROM as the representative product for the MCS-48 Family, we describe its functional blocks as follows.

1.1.1 Arithmetic Section

The arithmetic section of the processor contains the basic data manipulation functions of the 8048AH and can be divided into four blocks: Arithmetic Logic Unit (ALU), Accumulator, Carry Flag, and Instruction Decoder.

Instruction Decoder: The operation code (op code) portion of each program instruction is stored in the Instruction Decoder and converted to outputs which control the function of each of the blocks of the Arithmetic Section. These lines control the source of data and the destination register as well as the function performed in the ALU.

Accumulator: The accumulator is the single most important data register in the processor, being one of the sources of input to the ALU and often the destination of the result of operations performed in the ALU. Data to and from I/O ports and memory also normally passes through the accumulator.

Arithmetic Logic Unit: The ALU accepts 8-bit data words from one or two sources and generates an 8-bit result under control of the Instruction Decoder. The ALU can perform the following functions: Add with or without Carry; AND, OR, Exclusive OR. Increment/ Decrement; Bit Complement; Rotate Left, Right; Swap Nibbles; and BCD Decimal Adjust.

1.1.2 Program Memory

Resident program memory consists of 1024, 2048, or 4096 words with the width of eight bits which are addressed by the program counter. In the 8748H and the 8749H, this memory is user programmable and erasable EPROM; in the 8048AH/8049AH/8050AH, the memory is ROM which is mask programmable at the factory. The 8035AHL/8039AHL/8040AHL has no internal program memory and is used with external memory devices. Program code is completely interchangeable among the various versions. To access the upper 2K of program memory in the 8050AH, and other MCS-48 devices, a select memory bank and a JUMP or CALL instruction must be executed to cross the 2K boundary. There are three locations in Program Memory of special importance: Location 0, Location 3, and Location 7.

Location 0: Activating the Reset line of the processor causes the first instruction to be fetched from Location 0.

Location 3: Activating the Interrupt input line of the processor (if interrupt is enabled) causes a jump to subroutine at location 3.

Location 7: A timer/counter interrupt resulting from timer counter overflow (if enabled) causes a jump to subroutine at Location 7. Therefore, the first instruction to be executed after initialization is stored in Location 0, the first word of an external interrupt service routine is stored in Location 3, and the first word of a timer/counter service routine is stored at Location 7. Program memory can be used to store constants as well as program instructions.

1.1.3 Data Memory

Resident data memory is organized as 64, 128, or 256 by 8-bits wide in the 8048AH, 8049AH, and 8050AH. All locations are indirectly addressable through either of two RAM Pointer Registers which reside at address 0 and 1 of the register array. In addition, the first 8 location (0-7) of the array are designated as working registers and are directly addressable by several instructions. Since these registers are more easily addressed, they are usually used to store frequently accessed intermediate results. The DJNZ instruction makes very efficient use of working registers as program loop counters by allowing the programmer to decrement and test the register in a single insgwuction.

1.1.4 Input/Output

The 8048AH has 27 lines which can be used for input or output functions. These lines are grouped as 3 ports of 8 lines each serve as either inputs, outputs or bidirectional ports and 3 "test" inputs which can alter program sequences when tested by conditional jump instructions.

Ports 1 and 2: Port 1 and 2 are each 8 bits wide and have identical characteristics. Data written to these ports is statically latched and remains unchanged until rewritten. As input ports, these lines are non-latching. The lines of ports 1 and 2 are called quasi-bidirectional because of a special output circuit structure which allows each line to serve as an input, and output, or both even though outputs are statically latched.

Bus: Bus is also an 8-bit port which is a true bidirectional port with associated input and output strobes. If the bidirectional feature is not needed, Bus can serve as either a statically latched output port or non-latching input port. Input and output lines on this port cannot be mixed however.

1.1.5 Test and INT Inputs

Three pins serve as inputs and are testable with the conditional jump instruction. These are T0, T1, and INT. These pins allow inputs to cause program branches without the necessity to load an input port into the accumulator. The T0, T1, and INT pins have other possible functions as well.

1.1.6 Program Count and Stack

The program Counter is an independent counter while the Program Counter Stack is implemented suing pairs of registers in the Data Memory Array. Only 10,11, or 12 bits of the Program Counter are used to address the 1024, 2048, or 4096 words of on-board program memory of the 8048AH, 8049AH, or 8050AH, while the most significant bits can be used for external Program Memory fetches.

1.1.7 Program Status Word

An 8-bit status word which can be loaded to and from the accumulator exists called the Program Status Word (PSW). The PSW bit definitions are as follows:

Bits 0-2:	Stack Pointer bits(S0,S1,S2)
Bit 3:	Not used ("1" level when read)
Bit 4:	Working Register Bank Switch Bit (BS)
	0 = Bank 0; and 1 = Bank 1
Bit 5:	Flag 0 bit (F0) user controlled flag which can be complemented or cleaned, and tested with the conditional jump instruction JF0.

Bit 6: Auxiliary Carry (AC) carry bit generated by an ADD instruction and used by the decimal adjust instruction DAA.

Bit 7: Carry (CY) carry flag which indicates that the previous operation has resulted in overflow of the accumulator.

1.1.8 Conditional Branch Logic

The conditional branch logic within the processor enables several conditions internal and external to the processor to be tested by the users program. By using the conditional jump instruction, the conditions can effect a change in the program execution sequence.

1.1.9 Interrupt

An interrupt sequence is initiated by applying a low "0" level input to the INT pin. Interrupt is level triggered and active low to allow "WIRE ORing" of several interrupt sources at the input pin.

Interrupt Timing: The interrupt input may be enabled or disabled under Program Control using the EN I and DIS I instructions. Interrupts are disabled by Reset and remain so until enabled by the users program. An interrupt request must be removed before the RETR instruction is executed upon return from the service routine, otherwise, the processor will re-enter the service routine immediately. Many peripheral devices prevent this situation by resetting their interrupt request line whenever the processor accesses (Reads or Writes) the peripherals data buffer register. If the interrupting device does require access by the processor, one output line of the 8048AH may be designated as an "interrupt acknowledge" which is activated by the service subroutine to reset the interrupt request.

1.1.10 Timer/Counter

The 8048AH contains a counter to aid the user in counting external events and generating accurate time delays without placing a burden on the processor for these functions. In both modes, the counter operation is the same, the only difference being the source of the input to the counter.

Counter: The 8-bit binary counter is presettable and readable with two MOV instructions which transfer the contents of the accumulator to the counter and vice versa. The counter content may be affected by Reset and should be initialized by software. The increment from maximuı. count to zero (overflow) results in the setting of an overflow flag flip-flop and in the generation of an interrupt request. If timer and external interrupts occur simultaneously, the external source will be recognized and the Call will be to location 3.

As an Event Counter: Ex. cution of a START CNT instruction connects the T1 input pin to counter input and enables the counter. The T1 input is sampled at the beginning of state 3 or in later MCS-48 devices in state time 4. Subsequent high to low transitions on T1 will cause the counter to increment. T1 must be held low for at least 1 machine cycle to insure it won't be missed. T1 input must remain high for at least 1/5 machine cycle after each transition.

As a Timer: Execution of a START T instruction connects an internal clock to the counter input and enables the counter. The internal clock is derived by passing the basic machine cycle clock through a /32 prescaler. The prescaler is reset during the START T instruction. The resulting clock increments the counter every 32 machine cycles. Various delays from 1 to 256 counts can be obtained by presetting the counter and detecting overflow. Times longer than 256 counters may be achieved by accumulating multiple overflows in a register under software control. For time resolution less than 1 count, an external clock can be applied to the T1 input and the counter operated in the event counter mode. ALE divided by 3 or more can serve as this external clock. Very small delays or "fine turning" of larger delays can be easily accomplished by software delay loops.

1.1.11 Clock and Timing Circuits

Timing generation for the 8048AH is completely selfcontained with the exception of a frequency reference which can be XTAL, ceramic resonator, or external clock source. The Clock and Timing circuitry can be divided into the following three functional blocks:

Oscillator: The on-board oscillator is a high gain parallel resonant circuit with a frequency range of 1 to 11 MHz. If an accurate frequency reference is not required, ceramic resonator may be used in place of the crystal.

State Counter: The output of the oscillator is divided by 3 in the State Counter to create a clock which defines the state times of the machine (CLK). CLK can be made available on the external pin T0 by executing an ENTO CLK instruction. The output of CLK on T0 is disabled by Reset of the processor.

Cycle Counter: CLK is then divided by 5 in the Cycle Counter to provide a clock which defines a machine cycle consisting of 5 machine states.

1.1.12 Reset

The reset input provides a means for initialization for the processor. This Schmitt-trigger input has an internal pull-up device which in combination with an external 1 u fd capacitor provides an internal reset pulse of sufficient length to guarantee all circuitry is reset. Reset performs the following functions:

1. Sets program counter to zero;
2. Sets stack pointer to zero;
3. Selects register bank 0;
4. Selects memory bank 0;
5. Sets BUS to high impedance state (except when EA=5V);
6. Sets ports 1 and 2 to input mode;
7. Disables interrupts(timer and external);
8. Stops timer;
9. Clears timer flag;
10. Clears F0 and F1;
11. Disables clock output from T0.

1.1.13 Single-Step

This feature provides the user with a debug capacity in that the processor can be stepped through the program one instruction at a time. While stopped, the address of the next instruction to be fetched is available concurrently on BUS and the lower half of Port 2. The user can therefore follow the program through each of the instruction steps.

1.1.14 Power Down Mode

Extra circuitry has been added to the 8048AH/8049AH/8050AH ROM version to allow power to be removed from all but the data RAM array for low power standby operation. In the power down mode the contents of data RAM can be maintained while drawing typically 10% to 15% of normal operating power requirements.

1.1.15 External Access Mode

Normally the first 1K (8048AH), 2K (8949AH), or 4K (8050AH) words of program memory are automatically fetched from internal ROM or EPROM, the EA input pin however allows the user to effectively disable internal program memory by forcing all program memory fetches to reference external memory. The External Access mode is very useful in system test and debug because it allows the user to disable his internal applications program and substitute an external program of his choice -- a diagnostic routine for-instance.

1.1.16 Sync Mode

The 8048AH, 8049AH, and 8050AH has incorporated a new SYNC mode. The Sync mode is provided to ease the design of multiple controller circuits by allowing the designer to force the device into known phase and state time.

1.2 MCS-48 Expanded System

If the capabilities resident on the single-chip 8048AH/8748H /8035AHL /8049AH/ 8749H/8039AHL are not sufficient for your system requirements, special on-board circuitry allows the addition of a wide variety of external memory, I/O, or special peripherals you may require. The processors can be directly and simply expanded in the following areas:

a. Program Memory to 4K words;

b. Data Memory to 320 words (384 words with 8049AH);

c. I/O by unlimited amount; and

d. Special Functions using 8080/8085AH peripherals.

By using bank switching techniques, maximum capability is essentially unlimited. Expansion is accomplished in two ways:

a. Expander I/O A special I/O Expander circuit, the 8243, provides for the addition of four 4-bit Input/Output ports with the sacrifice of only the lower half (4 bits) of port 2 for inter-device communication. Multiple 8243's may be added to this 4-bit bus by generating the required "chip select" lines.

b. Standard 8085 Bus One port of the 8048AH/8049AH is like the 8-bit bidirectional data bus of the 8085 microcomputer system allowing interface to the numerous standard memories and peripherals of the MCS-80/85 microcomputer family.

1.3 MCS-48 Instruction Set

The MCS-48 instruction set is extensive for a machine of its size and has been tailored to be straightforward and very efficient in its use of program memory. All instructions are either one or two bytes in length and over 80% are only one byte long. Also, all instructions execute in either one or two cycles and over 50% of all instructions execute in a single cycle. Double cycle instructions include all immediate instructions, and all I/O instructions. The MCS-48 microcomputers have been designed to handle arithmetic operations efficiently in both binary and BCD as well as handle the single-bit operations required in control applications. Special instructions have also been included to simplify loop counters, table look-up routines, and N-way branch routines.

The MCS-48 instructions include the following functions:

1. Data Transfers;	2. Accumulator Operations;
3. Register Operations;	4. Flags;
5. Branch Instructions;	6. Subroutines;

7. Timer Instructions; 8. Control Instructions; and

9. Input/Output Instructions.

2. MCS-51 FAMILY

The MCS-51 family typified by the 8051 is a good example of an advanced 8-bit microcontroller. This family has been designed mainly for sequential control applications. There are three basic members of the MCS-51 family: the 8051, the 8031, and the 8751.
Three new devices -- the 8052, the 8032, and the 8752 -- are expanded versions with 8K of ROM, 256 bytes of RAM, and three timers. In addition, there are low-power CMOS versions designated 80C51, 80C31, and 87C51.

2.1 MCS-51 Microcontrollers Architectural Overview

2.1.1 Introduction

The 8051 is the original member of the MCS-51 family, and is the core for all MCS-51 devices. The features of the 8051 core are as follows:

1. 8-bit CPU optimized for control applications;
2. 64K Program Memory address space;
3. 64K Data Memory address space;
4. 4K bytes of on-chip Program Memory;
5. 128 bytes of on-chip Data RAM;
6. 32 bidirectional and individually addressable I/O lines;
7. two 16-bit counter/timers;
8. 6-source/5-vector interrupt structure with two priority levels;
9. Full duplex UART;

10. Extensive Boolean processing (single-bit logic) capabilities;

11. On-chip clock oscillator.

The 8031 has no on-board ROM and uses external memory for program storage. The 8751 is the same as the 8051 except that the ROM is replaced by UVEPROM (ultraviolet light erasable-programmable read only memory). The 8751 is relatively expensive and is meant to be a program development tool, to be replaced in production by the 8051 containing factory-masked ROM.

The basic architectural structure of the 8051 core is described as follows.

2.1.2 Memory Organization in MCS-51 Devices

Logical Separation of Program and Data Memory: All MCS-51 devices have separate address spaces for program storage (usually ROM) and data storage ((RAM). The logical separation of Program and Data Memory allows the Data Memory to be accessed by 8-bit addresses, which can be more quickly stored and manipulated by an 8-bit CPU.

Nevertheless, 16-bit Data Memory addresses can also generated through the DPTR register. Program Memory can only be read, not written to. There can be up to 64K bytes of Program Memory. In the ROM and EPROM versions of MCS-51 devices, the lowest 4K, 8K or 16K bytes of Program Memory are provided on-chip. In the ROMless versions, all Program Memory is external. The read strobe for external Program Memory is the signal PSEN (Program Strobe Enable). Data Memory occupies a separate address space from Program Memory. Up to 64K bytes of external RAM can be addressed in the external Data Memory space. The CPU generates read and write signals, RD and WR, as needed during external Data Memory accesses. External Program Memory and external Data Memory may be combined if desired by applying the RD and PSEN signals to the inputs of an AND gate and using the output of the gate as the read strobe to the external Program/Data memory.

Program Memory: After reset, the CPU begins execution from 0000H. The physical location of address 0000H is either on chip or external, depending on the 8051 pin designated EA (external address). If EA is low, address 0000H and all other program storage addresses will reference external memory. If EA is high, addresses 0000H to 0FFFH (to 1FFFH for 8052) will reference on-chip ROM; higher addresses will automatically reference external memory. The 8031 must operate with EA connected low because it does not contain any on-chip ROM. Some of the I/O pins are used for address and data when using external memory. Each interrupt is assigned a fixed location in Program Memory, the first being address 0003H. The interrupt causes the CPU to jump to that location, where it commences execution of the service routine. If the interrupt is not going to be used, its service location is available as general purpose Program Memory.

The interrupt service locations area spaced at 8-byte intervals. If an interrupt service routine is short enough (as is often the case in control applications), it can reside entirely within that 8-byte interval. Longer service routines can use a jump instruction to skip over subsequent interrupt locations, if other interrupts are in use. Program Memory addresses are always 16 bits wide, even though the actual amount of Program Memory used may be less than 64K bytes. External program execution sacrifices two of the 8-bit ports, P0 and P2, to the function of addressing the Program Memory.

Data Memory: There are the internal and external Data Memory spaces available to the MCS-51 user. There can be up to 64K bytes of external Data Memory. External Data Memory addresses can be either 1 or 2 bytes wide. One-byte addresses are often used in conjunction with one or more other I/O lines to page the RAM. Two-Two-byte addresses can also be used, in which case the high address byte is emitted at Port 2. Internal Data Memory space is divided into three blocks, which are generally referred to as the Lower 128 bytes, the upper 128 bytes (for 8052), and SFR (special function registers) space. The Lower 128 bytes of RAM are present in all MCS-51 devices. The lower 32 bytes are grouped into 4 banks of 8 registers. Program instructions call out these registers as R0 through R7. Two bits in the Program Status Word (PSW) select which register bank is in use. Only one bank at a time can be in active use. This allows more efficient use of code space, since register instructions are shorter than instructions that use direct addressing. The next 16 bytes above the register banks form a block of bit-addressable memory space. All of the bytes in the Lower 128 can be accessed by either direct or indirect addressing. The Upper 128 can only be accessed by indirect addressing. The Upper 128 bytes of RAM are not implemented in the 8051, but are in the devices with 256 bytes of RAM.

2.2 MCS-51 SFR Space: Special Function Registers

The registers associated with important functions of the 8051 are assigned memory locations in the on-chip data storage space, allowing them to be addressed by program instructions. Some of the SFR locations are bit addressable as well as byte addressable. This feature is not found in processors such as 8085. Not all the SFR addresses are occupied. Unoccupied addresses in SFR space are reserved for use in future versions of the 8051 and should not be used in programs for current version.

Accumulator(ACC) and B Register: When referring to the accumulator as a location in the SFR, the mnemonic ACC is used. Accumulator-specific instructions designate the accumulator as A. The accumulator in the 8051 has the same functions as accumulator in processors such as the 8085. It is also used in some instructions as an index register. The B register has a specific function in multiply and divide operations. Otherwise, it can be used as a general-purpose scratchpad register.

Program Status Word (PSW): The PSW contains several status bits that reflect the current state of the CPU. The PSW resides in SFR space. It contains the Carry bit, other than serving the functions of a Carry bit in arithmetic operations, it also serves as the "Accumulator" for a number of Boolean operations; the Auxiliary Carry (for BCD operations); the two register bank select bits RS0 and RS1, a number of instructions refer to these RAM locations as R0 through R7, the selection of which of the four banks is being referred to is made on the basis of the bits RS0 and RS1 at execution time; the Overflow flag; a Parity bit, which reflects the number of 1s in the Accumulator: P =1 stands for odd number of 1s, P = 0 for even number of 1s, thus the number of 1s in the Accumulator plus P is always even; and two user-definable status flags.

Stack Pointer (SP): Since SP is 8-bit wide, it allows a maximum stack size of 256 bytes. In contrast to the 8085, the stack in the 8051 grows upward through memory; hence, the SP is incremented before data are stored as a result of a PUSH or CALL instruction. The stack may reside anywhere in on-chip RAM by loading the appropriate address into the SP After reset, the SP contains the address 07H, causing the stack to start at location 08H in a register bank. Because the program typically has other uses for the register banks, the stack is usually moved higher in RAM by loading a new address into SP before doing any PUSH or CALL instructions.

Data Pointer (DPTR): The DTPR is a 16-bit quantity held in two 8-bit parts: the high byte in DPH and the low byte in DPL. The main purpose of the DPTR is to hold a 16-bit address for certain instructions. It can be used as a single 16-bit register or as two 8-bit registers.

Port Latches (P0,P1,P2,P3): The 32 I/O pins are organized into four 8-bit ports designated P0 - - P3. Each port has an associated 8-bit latch, the outputs of which drive the matching I/O pins. The contents of the latches can be read from or written to in the SFR.

Serial Data Buffer (SBUF): The SBUF is actually two separate registers sharing a common address: One is read-only and the other write-only. When data are written to SBUF, they go to a transmit buffer and are held there for serial transmission. When data are read from SBUF, they come from the serial data receive buffer.

Timer Registers: Registers TH0 and TL0 are the high and low bytes, respectively, of 16-bit counting register for timer/counter 0. Likewise, TH1 and TL1 are for timer/counter 1. In the 8052, TH2 and TL2 are for timer/counter 2. Also the 8052 contains two 8-bit capture registers (RCAP2H and RCAP2L) used to hold copies of the TH2 and TL2 register contents.

Control Registers: The SFR contains registers used for the control and status of the interrupt system, the timer/counters, and the serial port. They are IP (interrupt priority), IE (interrupt enable), TMOD (timer mode), TCON (time control), T2CON (8052 timer 2 control), SCON (serial port control), and PCON (power control, used mainly in 80C51).

2.3 MCS-51 I/O Ports

One of the most useful features of the 8051 is the I/O, consisting of four bidirectional ports. Each port has an 8-bit latch in the SFR space, an output driver, and an input buffer. The ports can be used for general I/O, as address and data lines, and for certain spacial functions.

Input, Loading, and Output Drive: Ports 1,2, and 3 have the equivalent of internal pull-up resistors. When used as inputs, the pins of P1, P2, and P3 will be high (logic 1) when open-circuited and will source current when pulled low by an external device. Port 0 does not have the same pull-up feature and is floating (high impedance) when used as an input. A clarification is needed for the phrase "read a port." Some instructions read the actual level on the I/O pin; others read the level in the latch. A condition can occur where the port is being used to output a high (logic 1), but the external load attached to the port is of such low resistance that the voltage on the pin is at the same level as a low (logic level 0). Reading the latch would a 1, but reading the pin would show a 0. Instructions that do a read-modify-write operation(e.g., INC) read the latch. When used as outputs, ports 1, 2, and 3 each can drive the equivalent of four LS TTL inputs. Port 0 can drive eight such equivalent inputs. To speed up 0 to 1 output transitions (i.e., to overcome capacitance effects) on ports 1, 2, and 3, an additional internal pull-up is activated briefly during output.

Alternate Port Functions: All the pins of port 3 (and in the 8052, two P1 pins) have an alternate function. To enable an alternate function, a 1 must be written to the corresponding bit in the port latch.

Accessing External Memory: Since 8051 has separate program memory and data memory, it uses different hardware signals to access the corresponding external storage devices. The PSEN (program strobe enable) signal is used as the read strobe for program memory, and RD and WR are used as the read and write strobes to access data memory. Accesses to external program memory always use a 16-bit address. Accesses to external data memory may use either an 8-bit or a 16-bit address, depending on the instruction being executed. In the case of 16-bit address, the high-order 8 bits of the address are output on port 2, where they are held constant during the entire memory access cycle. The prior contents of the port 2 latches in the SFR are not lost but are restored after the memory access cycle. If an 8-bit address is being used, the contents of port 2 are unchanged, which allows some of the port 2 pins to be used to select 256-byte pages for the lower 8 bits of the address. The low-order 8 bits of the address are multiplexed with the data byte on port 0. When used in this mode, the port 0 pins are connected to an internal active pull-up; they do not float. The prior contents of the port 0 latches are lost. The ALE (address latch enable) signal must be used to capture the low-order address bits in an external latch.

2.4 MCS-51 Timer/Counters

The 8051 has two 16-bit registers that can be used as either timers or counters. They are designated timer 0 and timer 1. The 8052 has an additional 16-bit register designated timer 2. These registers are in the SFR as pairs of 8-bit registers. When used as a timer, the register is incremented once per machine cycle, which is equal to once per 12 clock periods. When used as a counter, the register is incremented on a 1 - 0 transition (a negative edge) applied to the appropriate input pin: T0 or T1 (or T2 in the 8052). It takes two complete machine cycles for the 8051 to see 1 - 0 transition; the input must be held high for at least one cycle and then low for at least one cycle.

Timer 0 and Timer 1: The way that timer 0 and timer 1 will operate is determined by the 8 bits written to the TMOD register. The bits M0 and M1 (a pair for each timer) are used to select one of four operating modes: mode 0, mode 1, mode 2, and mode 3. Both counters work the same in mode 0,1, and 2, but different in mode 3.

Timer 2: Timer 2 is a 16-bit timer/counter in the 8052 group of devices; it does not appear in the 8051 group. The input source for timer 2 can be the clock (timer operation) or an external input (counter operation). Timer 2 has three operating modes: capture, auto-load, and baud rate generator. The input source and operating mode are selected by bits in the T2CON control register.

2.5 MCS-51 Serial Port Interface

The 8051 has a full duplex serial port that allows data to be transmitted and received simultaneously in hardware while the software is doing other things. A serial port interrupt is generated by the hardware to get the attention of the program in order to read or write serial port data. The receiver hardware is double buffered, meaning that a received frame of data can be held for reading while a second frame is being received. Double buffering allows the receiver interrupt service routine to be less time critical, but the stored frame must be read before reception of the second frame is complete or the stored frame will be overwritten and lost. To obtain the same level of performance with a processor such as the 8085, the use of a separate USART (universal synchronous/asynchronous receiver/transmitter) chip is required. Both the transmit and receive buffers are accessed at the same location in the SFR space: the SBUF register. Writing to SBUF loads the transmit buffer, and reading from SBUF obtains the contents of the receive buffer. Any instruction that writes to SBUF initiates serial transmission. The serial port has four modes of operation: mode 0, 1, 2, and 3. They are not to confused with the timer/counter modes.

Serial Port Control Register (SCON): SCON is the serial port control and status register. Bits SM0 and SM1 are used to select the operating mode. The SM2 bit is used in a multiprocessor system where one 8051 acts as a master unit, sending commands to one or more slave units.

Serial Port Modes: Mode 0 is half-duplex synchronous operation. Data are sent and received (not simultaneously) through the RXD pin in 8-bit frames, LSB first. The bit rate is fixed at one-twelfth the oscillator frequency. The shift clock which is the same frequency as the bit rate is sent out the TXD pin during both transmission and reception and is used to synchronize the receiver to the sender.Reception is initiated when bit REN is set to 1 and bit RI is cleared to 0 in the SCON register.

Mode 1 is full duplex asynchronous operation. Data are sent out TXD and received through RXD. A complete frame consists of a start bit (always a 0), followed by 8 data bits (LSB first), followed by a stop bit (always a 1). The start and stop bits are added by the hardware; the software writes the 8-bit data byte to, or reads from, SBUF. The baud rate is variable and can be obtained by using timer 1 as a baud rate generator.

Mode 2 is similar to mode 1, with two exceptions. First, the frame is 11 bit long; a ninth data bit is inserted before the stop bit. When transmitting, the ninth bit is obtained from TB8 in SCON. It assumed that TB8 was written into before initiating transmission. When receiving, the ninth bit can be read from RB8 in SCON. A common use for the ninth bit is as a parity bit for 8-bit data. The second difference is that the baud rate is either 1/32nd or 1/64th of the oscillator frequency, as selected by the SMOD bit (bit7) in the PCON register. If SMOD is set to 1, the 1/32 number is used. If SMOD is cleared to 0, 1/64 is used.

Mode 3 is the same as mode 2, except that the baud rate is variable and can be obtained in the same way as in mode 1. Reception for modes 1,2, and 3 is enabled when the REN bit in SCON is set to 1. Actual reception is initiated when an incoming start bit causes a high to low transition on the RXD pin. Transmission is initiated by writing to SBUF in any serial mode.

Timer 1 as Baud Rate Generator: Although it is possible to use the timer/counter as a baud rate generator in any of its modes, its most common use is as a timer (i.e., clock-sourced) in auto-reload mode (timer mode 2). The baud rate is then given by:

Baud Rate = Oscillator Frequency / N*(256 - TH1)

where N depends on the SMOD bit in the PCON register (bit 7). If SMOD = 0, N = 384. If SMOD = 1, N = 192. TH1 represents the contents of register TH1.

The Power Control Register (PCON): Of the 8 bits in the PCON, only bit 7, SMOD, is implemented in the standard 8051. SMOD is used in setting the baud rate of the serial port. Bits 0, 1, 2, and 3 are implemented in the CMOS version. Bits 0, and 1 are used in power saving modes, and bits 2, and 3 are general-purpose flags.

2.6 MCS-51 Interrupts

The 8051 has five sources of interrupts which are associated with bit locations in registers: two from external pins (INT0 and INT1), two from the timer/counters (TF0 and TF1), and one from the serial port (TI or SI). These associated bits can be set or cleared by software, with the same results as when the bits are set or cleared by hardware. All interrupts or each individual interrupt can be enabled or disabled by setting or clearing the appropriate bit in the IE register. If enabled, an interrupt will cause a call to one of the predefined locations in RAM. The return address is automatically pushed onto the stack before jumping and is popped back off when an

RETI (return from interrupt) instruction is execution is executed. If an interrupt occurs while it is disabled, or while a higher priority one is running, it becomes pending. As soon as a pending interrupt is enabled, it will cause a call, unless it was canceled by software while it was still pending or a higher priority interrupt was simultaneously made active.

Servicing External Devices: The timing of events in external devices usually has no relationship to the timing of the CPU; that is, real-world events are usually asynchronous to the processor. In order to monitor and control external devices, a microcomputer must have a method of responding to I/O requests and other external events in a timely manner.

Polling and Buffering: One way a processor deals with I/O devices is to ask them periodically if they need service, that is to poll them. Often, a polled device will exchange blocks of information with the processor. The device will hold such blocks in its own memory, the buffer. The main drawback of polling is the amount of time the CPU spends checking the I/O device. If the I/O device buffer either fills up or empties out while it is waiting for service, data may be lost or the device may stop working. Consider a printer, the processor initially will load the printer buffer with text. If the buffer runs out before the printer is polled again, it will stop. For digital communication equipment, if data keep coming in, the receiver buffer can overflow while waiting for the next poll. Real-time applications which require service from the processor as soon as the need occurs cannot wait for a poll. Therefore, something better is called for, it is an interrupt.

Basic Interrupt Action: I/O devices often require immediate service while the processor is in the middle of doing something else. The interrupt is a software-controlled hardware feature that, in an orderly manner, forces the processor to suspend what it is currently doing in order to service the I/O device.

When it is finished with I/O, the processor will resume where it left off, much the same as a subroutine. Interrupts that can be blocked by software are called maskable; those that cannot be blocked are called nonmaskable. Interrupts that occur while masked are called to be pending. An I/O device will request service by activating an interrupt pin on the CPU. If the CPU has enabled its interrupts in software, it will initiate its response, often with an acknowledgment signal to the I/O device. This request-acknowledge sequence is an example of handshaking. The rest of the response is similar to a subroutine call. The CPU will push the return address onto the stack and branch to a predefined part of memory, where it expects to find the interrupt service routine which will handle the interrupting device. The last instruction in the routine will be a RETURN, which will pop the return address off the stack and into the PC register. The CPU will then resume program execution from the point where it was interrupted.

Multiple Interrupt Sources and Vectoring: When an interrupt occurs, the processor must first determine which device was the source, as different devices need different services. One way is for the interrupt service routine to poll all the devices to find out which one called, perhaps by having the service routine read status lines from all the devices. But polling can be time-consuming. A faster way is for each interrupting device to point (like a vector arrow) to the place in memory where its service routine is stored. The CPU can then do there directly. Such a system is called a vectored interrupt. In a vectored system, a problem will occur if two devices request an interrupt at exactly the same time. It needs a means of establishing priority.

Priority Levels: The 8051 has a two-tier priority structure. The top tier has two priority levels: high and low. Each interrupt source can be assigned to either high-level or low-level status by setting the appropriate bits in the IP register. When two interrupts of different levels are received simultaneously, the higher level interrupt is serviced first. The second tier of priority is used to resolve simultaneous interrupts within the same level. The priority-within-level ordering from highest to lowest is fixed as: IE0, TF0, IE1, TF1,RI, OR TI.

Interrupt Timing and Handing: A "snap-shot" (sample) of the interrupt flags is taken by the 8051 hardware at the end of a typical machine cycle(C1). During the following machine cycle(C2), the sample from the previous cycle is examined. If one of the flags sampled during C1 is found to be set, then a call to the appropriate interrupt vector will be generated during cycles C3 and C4. Execution of the interrupt service routine will start with cycle C5 and continue for as long as is required by the routine. The hardware will not generate the interrupt call if one of the following is true:

1. An interrupt of equal or higher priority is already in progress.
2. The current machine cycle is not the final cycle of the instruction being executed.
3. The instruction in progress is RETI or any instruction that writes to the IE or IP register.

The time between the activation of an interrupt and the start of execution of the service routine is the response time. In the 8051, the shortest response time is three machine cycles and the longest (worst case) is nine machine cycles.

Activation Levels and Flag Clearing: Interrupts can be either level-activated or transition-activated (level-triggered or edge-trigged). Because a transition-activated event is, by definition, a transient, when the flag bit associated with the interrupt is cleared, the interrupt event itself is cleared. On the other hand, clearing the flag bit of a level-activated interrupt will have no effect if the external level causing the interrupt stays active.

2.7 MCS-51 Instructions and Addressing

The 8051 has five addressing modes:

1. Direct Addressing;
2. Indirect Addressing;
3. Register Instructions;
4. Immediate Operand Instructions; and
5. Indexed Addressing.

The 8051 assembly language uses two special symbols: @, and # to distinguish operand types. The @ before an operand means that indirect addressing is being used; the # before an operand means it is an immediate operand (a constant). The 8051 has five groups of instructions consisting of 111 instruction types: 49 one-byte, 45 two-byte, and 17 three-byte as follows:

1. Arithmetic Operations;
2. Logic Operations;
3. Data Transfers;
4. Boolean Operations; and
5. Branching Instructions.

3. MCS-80/85 Family

3.1 8080A 8-bit N-Channel Microprocessor

The Intel 8080A is a complete 8-bit parallel central processing unit (CPU). It is fabricated on a single LSI chip using Intel's n-channel silicon gate MOS process. This offers a the user a high performance solution to control and processing applications. It contains 6 8-bit general purpose working registers and an accumulator. These six registers may be addressed individually or in pairs providing both single and double precision operators. Arithmetic and logical instructions set or reset 4 testable flags. A fifth flag provides decimal arithmetic operation.

The 8080A has an external stack feature wherein any portion of memory may be used as a last in/first out stack to store/receive the contents of the accumulator, flags, program counter, and all of the 6 general purpose registers. The 16-bit stack pointer controls the addressing of this external stack. This stack gives the 8080A the ability to easily handle multiple level priority interrupts by rapidly storing and restoring processor status. It also provides almost unlimited subroutine nesting. This microprocessor has been designed to simplify systems design. Separate 16-line address and 8-line bidirectional data busses are used to facilitate easy interface to memory and I/O. Singles to control the interface to memory and I/O are provided directly by the 8080A.

Ultimate control of the address and data busses reside with the HOLD signal. It provides the ability to suspend processor operation and force the address and data busses into a high impedance state. This permits OR-tying these busses with other controlling devices for DMA or multi-processor operation.

Instruction Set: The accumulator group instructions include arithmetic and logical operators with direct, indirect, and immediate addressing modes. Move, load, and store instruction groups provide the ability to move either 8 or 16 bits of data between memory, the six working registers and the accumulator using direct, indirect, and immediate addressing modes. The ability to branch to different portions of the program is provided with jump, jump conditional, and computed jumps. Also the ability to call to and return from subroutines is provided both conditionally and unconditionally. The RESTART (or single byte call instruction) is useful for interrupt vector operation.

Double precision operators such as stack manipulation and double add instructions extend both the arithmetic and interrupt handling capability of the 8080A. The ability to increment and decrement memory, the six general purpose registers and the accumulator is provided as well as extended increment and decrement instructions to operate on the register pairs and stack pointer. Further capability is provided by the ability to rotate the accumulator left or right through or around the carry bit.

Input and output may be accomplished using memory addresses as I/O ports or the directly addressed I/O provided for in the 8080A instruction set. The following special instruction group completes the 8080A instruction set: the NOP instruction, HALT to stop processor execution and the DAA instructions provide decimal arithmetic capability. STC allows the carry flag to be directly set, and the CMC instruction allows it to be complemented. CMA complements the contents of the accumulator and XCHG exchanges the contents of two 16-bit register pairs directly.

3.2 8085AH 8-bit HMOS Microprocessors

The Intel 8085AH is a complete 8-bit parallel Central Processing Unit (CPU) implemented in N-channel, depletion load, silicon gate technology (HMOS). Its instruction set is 100% software compatible with the 8080A microprocessor, and it is designed to improve the present 8080A's performance by higher system speed. Its high level of system integration allows a minimum system of three IC's [8085AH(CPU), 8156H(RAM/IO) and 8755A(EPROM/IO] while maintaining total system expendability. It requires a single +5V supply. Its basic clock speed is 3 MHz(8085AH), 5 MHz(8085AH-2), or 6MHz(8085AH-1). The 8085AH-2 and 8085AH-1 are faster versions of the 8085AH. The 8085AH incorporates all of the features that taw 8224 (clock generator) and 8228(system controller) provided for the 8080A, thereby offering a higher level of system integration.

The 8085AH uses a multiplexed data bus. The address is split between the higher 8-bit Address Bus and the lower 8-bit Address/Data bus. During the first T state (clock cycle) of a machine cycle the low order address is sent out on the Address/Data Bus. These lower 8 bits may be latched externally by the Address Latch Enable (ALE) signal. ALE is used as a strobe to sample the lower 8-bits of address on the Data Bus. During the rest of the machine cycle the data bus is used for memory or I/O data. The on-chip address latches of 8155H/8156H/8755A memory products allow a direct interface with the 8085AH. It has 12 addressable 8-bit registers.

Four of them can function only as two 16-bit register pairs. Six others can be used interchangeably as 8-bit registers or 16-bit register pairs. They are: 8-bits Accumulator (ACC or A); 16-bit address Program Pointer (PC); 8-bits*6 or 16-bits*3 General Purpose Registers (BC,DE) and Data Pointer (HL); 16-bit address Stack Pointer (SP); and 5 8-bit space Flag Registers (Flags or F). It provides RD, WR, S0, S1, and IO/M signals for bus control. An interrupt Acknowledge signal (INTA) is also provided. HOLD and all interrupts are synchronized with the processor's internal clock. It also provides SID (Serial Input Data and SOD (Serial Output Data) lines for simple serial interface. The 8085AH has 5 interrupt inputs: INTR which is identical in function to the 8080A INT; three RESTART inputs: RST 5.5, RST

6.5, RST 7.5 among which each has a programmable mask; and a nonmaskable RESTART interrupt TRAP.

You may drive the clock inputs of the 8085AH, 8085AH-2, or 8085AH-1 with a crystal, an LC tuned circuit, an RC network, or an external clock source. The crystal frequency must be at least 1 MHz, and must be twice the desired internal clock frequency; hence the 8085AH is operated with a 6 Mhz crystal (for 3 MHz clock), the 8085AH-2 with a 10 MHz crystal (for 5 MHz clock), and the 8085AH-1 with a 12 MHz crystal (for 6 MHz clock). If your system requirements are such that slow memories or peripheral devices are being used, a wait state generating circuit may be used to insert one WAIT state in each 8085AH machine cycle. There are 7 possible types of machine cycles. Which of these seven takes place is defined by the status of the 3 status lines (IO/M, S1,S0) and the 3 control signals (RD, WR, and INTA).

3.3 8155H,8156H 2048-bit Static HMOS RAM with I/O Ports & Timer

The Intel 8155H and 8186H are RAM and I/O chips implemented in N-channel, depletion load, silicon gate technology (HMOS), to be used in the 8085AH and 8088 microprocessor systems. The RAM portion is designed with 2048 static cells organized as 256*8. They have a maximum access time of 400 ns to permit use with no wait states in 8085AH CPU. The 8155H-2 and 8256H-2 have maximum access times of 330 ns for use with the 8085AH-2 and the 5 MHz 8088 CPU. The I/O portion consists of three general purpose I/O ports. One of the three ports can be programmed to be status pins, thus allowing the other two ports to operate in handshake mode.

A 14-bit programmable counter/timer is also included on chip to provide either a square wave or terminal count pulse for the CPU system depending on timer mode. The 8055H/8156H contains three functional units:

1. 2K bit Static RAM organized as 256*8;
2. Two 8-bit I/O ports (PA & PB) and one 6-bit I/O port (PC); and
3. 14-bit timer-counter.

The IO/Memory select pin IO/M selects either the five registers (Command, Status, PA0-7, PB0-7, PC0-5) or the memory (RAM) portion. The 8-bit address on the Address/Data lines, Chip Enable input CE, and IO/M are all latched on-chip at the falling edge of ALE. The I/O section of the 8155H/8156H consists of five registers: Command/Status Registers(C/S); PA Register; PB Register; and PC Register. The time is a 14-bit down-counter that counts the TIMER IN pulses and provides either a square wave or pulse when terminal count (TC) is reached.

The command register consists of eight latches. Four bits (0-3) define the mode of the ports, two bits (4-5) enable or disable the interrupt from port C when it acts as control port, and the last two bits (6-7) are for the timer. The status register consists of seven latches, one for each bit; six (0-5) for the status of the ports and one (6) for the status of the timer.

3.4 8185/8185-2 1024 8-bit Static RAM for MCS-85

The Intel 8185 is an 8192-bit static random access memory (RAM) organized as 1024 words by 8-bits using N-channel Silicon-Gate MOS technology. The multiplexed address and data bus allows the 8185 to interface directly to the 8085AH and 8088 microprocessors to provide a maximum level of system integration. The low standby power dissipation minimizes system power requirements when the 8185 is disabled. The 8185-2 is a high-speed selected version of the 8185 that is compatible with the 5 MHz 8085 AH-2 and the 5 MHz 8088. The

8185 haws been designed to provide for direct interface to the multiplexed bus structure and bus timing of the 8085A microprocessor.

At the beginning of an 8185 memory access cycle, the 8-bit address on AD0-7, A8 and A9, and the status of CE1 and CE2 are all latched internally in the 8185 by the falling edge of ALE. If the latched status of both CE1 and CE2 are active, the 8185 powers itself up, but no action occurs until the CS line goes low and the appropriate RD or WR control signal input is activated. The CS input is not latched by the 8185 in order to allow the maximum amount of time for address decoding in selecting the 8185 chip. Maximum power consumption savings will occur, however, only when CE1 and CE2 are activated selectively to power down the 8185 when it is not in use. A possible connection would be to write the 8085A's IO/M line to the 8185's CE1 input, thereby keeping the 8185 powered down during I/O and interrupt cycles.

8224 Clock Generator and Driver for 8080A CPU: The Intel 8224 is a single chip clock generator/driver for the 8080A CPU. It is controlled by a crystal, selected by the designer to meet a variety of system speed requirements. It also includes circuits to provide power-up reset, advance status strobe, and synchronization of ready. It provides the designer with a significant reduction of packages used to generate clocks and timing for the 8080A.

3.5 8228 System Controller and Bus Driver for 8080A CPU

The Intel 8228 is a single chip system controller and bus driver for MCS-80. It generates all signals required to directly interface MCS-80 family RAM, ROM, and I/O components. A bidirectional bus driver is included to provide high system TTL fan-out. It also provides isolation of the 8080 data bus from memory and I/O. This allows for the optimization of control signals, enabling the systems design to use slower memory and I/O. The isolation of the bus driver also provides for enhanced system noise immunity.

A user selected single level interrupt vector (RST 7) is provided to simplify real time, interrupt driven, small system requirements. The 8228 also generates the correct control signals to allow the use of multiple byte instructions (e.g., CALL) in response to an interrupt acknowledge by the 8080A. This permits large, interrupt driven systems to have an unlimited number of interrupt levels. The 8228 is designed to support a wide variety of system bus structures and also reduce system package count for cost effective, reliable design of MCS-80 systems.

3.6 8755A 16,384-bit EPROM with I/O

The Intel 8755A is an erasable and electrically reprogrammable ROM (EPROM) and I/O chip to be used in the 8085AH microprocessor systems. The EPROM portion is organized as 2048 words by 8 bits. It has a maximum access time of 450 ns to permit use with no wait states in an 8085AH CPU. The I/O portion consists of 2 general purpose I/O ports. Each I/O port has 8 port lines, and each I/O port line is individually programmable as input or output.

PROM Section: The 8755A contains an 8-bit address latch which allows it to interface directly to MCS-48, and MCS-85 processors without additional hardware. The PROM section of the chip is addressed by the 11-bit address and the Chip Enables. The address, CE1 and CE2 are latched into the address latches on the falling edge of ALE. If the latched Chip Enables are active and IO/M is low when RD goes low, the contents of the PROM location addressed by the latched address are put out on the AD0-7 lines (provided that V^{DD} is tied to V^{CC}).

I/O Section: The I/O section of the chip is addressed by the latched value of AD0-1. Two 8-bit Data Direction Registers (DDR) in 8755A determine the input/output status of each pin in the corresponding ports.

4. Intel 80186/80188 Family

4.1 Intel 80186 High Integration 16-bit Microprocessor

The Intel 80186 ia a highly integrated 16-bit microprocessor. The 80186 effectively combines 15-20 of the most common 8086 system components onto one chip and provides two times greater throughput than the standard 5 MHz 8086. The 80186 is object code compatible with the 8086/8088 microprocessors and adds 10 new instruction types to the 8086/8088 instruction set.

4.1.1 80186 Base Architecture

The 8086,8088, 80186, and 80286 family all contain the same basic set of registers, instructions, and addressing modes.

Register Set: The 80186 base architecture has fourteen registers which are grouped into four categories:

1. **General Registers:** Eight 16-bit general purpose registers may be used for arithmetic and logic operands. Four of them (AX, BX, CX, and DX) can also be split into pairs of separate 8-bit registers.

2. **Segment Registers:** Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data.

3. **Base and Index Registers:** Four of the general registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing

mode selects the specific registers for operand and address calculations.

4. **Status and Control Registers:** Two 16-bit special purpose registers record or alter certain aspects of the 80186 processor state. These are the Instruction Pointer Register which contains the offset address of the next sequential instruction to be executed, and the Status Word Register which contains status and control flag bits.

Status Word Description: The Status Word records specific characteristics of the result of logical and arithmetic instructions (bits 0,2,4,6,7, and 11) and controls the operation of the 80186 within a given operating mode (bits 8,9, and 10).

Instruction Set: The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high-level instructions, and processor control. These categories can be summarized as follows: general purpose, input/output, address object, flag transfer, addition, subtraction, multiplication, division, logicals, shifts, rotates, flag operations, external synchronization, no operation, high level instructions, conditional transfers, unconditional transfers, iteration controls, and interrupts.

Addressing Modes: The 80186 provides eight categories of addressing modes to specify operands:

1. **Register Operand Mode:** The operand is located in one of the 8- or 16-bit general registers;

2. **Immediate Operand Mode:** The operand is included in the instruction;

3. **Direct Mode:** The operand's offset is contained in the instruction as an 8- or 16-bit displacement element.

4. **Register Indirect Mode:** The operand's offset is in one of the registers SI,DI BX, or BP.

5. **Based Mode:** The operand's offset is the sum of an 8- or 16-bit displacement and the contents of a base register (BX, or BP).

6. **Indexed Mode:** The operand's offset is the sum of an 8- or 16-bit displacement and the contents of an index register (SI or DI).

7. **Based Indexed Mode:** The operand's offset is the sum of contents of a base register and an index register.

8. **Based Indexed Mode with Displacement:** The operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement.

Data Types: The 80186 directly supports the following data types:

Integer, Ordinal, Pointer, String, ASCII, BCD, Packed BCD, Floating Point. In general, individual data elements must fit within defined segment limits.

I/O Space: The I/O space consists of 64K 8-bit or 32K 16-bit ports. Separate instructions address the I/O space with either an 8-bit port address specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A15-A8 LOW. I/O port addresses 00F8H through 00FFH are reserved.

Interrupts: An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Status Word) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated which occur in response to an external input and are classified as non-maskable or maskable; INT instructions with which programs may cause an interrupt; and instruction exceptions which occur when an unusual condition preventing further instruction processing is detected while attempting to execute an instruction.

4.1.2 80186 Clock Generator

The 80186 provides an on-chip clock generator for both internal and external clock generation. The clock generator features a crystal oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

Oscillator: The oscillator circuit of the 80186 is designed to be used with a parallel resonant fundamental mode crystal. It is used as the time base for the 80186. The crystal frequency selected will be double the CPU clock frequency.

Clock Generator: The 80186 Clock Generator provides 50% duty cycle processor clock for the 80186. It does this by dividing the oscillator output by 2 forming the symmetrical clock. If an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal.

READY Synchronization: The 80186 provides both synchronous and asynchronous ready inputs.

RESET Logic: The 80186 provides both a RES input pin and a synchronized RESET output pin for use with other system components.

4.1.3 Local Bus Controller

The 80186 provides a local bus controller to generate the local bus control signals. It also employs a HOLD/HLDA protocol for relinquishing the local bus to other bus masters, and provides outputs that can be used to enable external buffers and to direct the flow of data on and off the local bus.

Memory/Peripheral Control: The RD and WR signals are used to strobe data from memory or I/O to the 80186 or to strobe data from the 80186 to memory or I/O. The ALE line provides a strobe to latch the address when it is valid.

Transceiver Control: The 80186 generates two control signals, DT/R and DEN which are generated to control the flow of data through the transceivers, to be connected to transceiver chips. This capability allows the addition of transceivers for extra buffering without adding external logic.

Local Bus Arbitration: The HOLD/HLDA system of local bus exchange provides an asynchronous bus exchange mechanism. Multiple masters utilizing the same bus can operate at separate clock frequencies. External circuitry must arbitrate which external device will gain control of the bus when there is more than one alternate local bus master.

4.1.4 Internal Peripheral Interface

All the 80186 integrated peripherals are controlled by 16-bit registers contained within an internal 256-byte control block which may be mapped into either memory or I/O space. Internal logic will recognize control block addresses and respond to bus cycles. The control block base address is programmed by a 16-bit relocation register contained within the control block at offset FEH from the base address of the block.

Chip-Select/Ready Generation Logic: The 80186 contains logic which provides programmable chip-select generation for both memories and peripherals. It can be programmed to provide READY (or WAIT state) generation, and also latched address bits A1 and A2. The chip-select lines are active for all memory and I/O cycles in their programmed areas, whether they be generated by the CPU or by the integrated DMA unit.

Upper Memory CS, Lower Memory CS, and Mid-Range Memory CS: The chip select UCS is provided for the top of the memory which is usually used as the system memory because after reset the 80186 begins executing at memory location FFFF0H. The upper limit of memory defined by the UCS is always FFFFFH, while the lower limit is programmable which is defined in the UMCS register. The chip select LCS is provided for low memory. The bottom of memory contains the interrupt vector table starting at location 00000H. The lower limit of memory defined by LCS is always 0H while the upper limit which is defined in the LMCS register is programmable.

The four MCS lines used for mid-range memory are active within a user-locatable memory block defined by the MCS lines.

Peripheral Chip Selects: The 80186 can generate chip selects for up to seven peripheral devices. These chip selects called PCS0-6 are active for seven contiguous blocks of 128 bytes above a programmable base address which may be located in either memory or I/O space.

4.1.5 DMA Channels

The 80186 DMA controller provides two independent DMA channels. Data transfers can occur between memory and I/O spaces or within the same space. Data can be transferred either in bytes (8 bits) or words (16bits) to or from even or odd addresses. Each DMA channel maintains both a 20-bit source and destination pointer which can be optionally incremented or decremented after each data transfer. Each data transfer consumes 2 bus cycles (a minimum of 8 clocks), one cycle to fetch data and the other to store data. This provides a maximum data transfer rate of 1.25M Words/sec or 2.5 M Bytes/sec at 10 MHz.

DMA Operation: Each channel has six registers in the control block which define each channel's operation. The control registers consist of a 20-bit Source Pointer, a 20-bit Destination Pointer, each of these two pointers takes up two full 16-bit registers in the peripheral control block; a 16-bit Control Word which determines the mode of operation for the particular 80186 DMA channel. The Transfer Count Register (TC) which is decremented after every DMA cycle specifies the number of DMA transfers to be performed. Up to 64 K byte or word transfers can be performed with automatic termination. All registers may be modified or altered during any DMA activity. Any changes to these registers will be reflected immediately in DMA operation.

DMA Requests: Data transfers may be either source or destination synchronized, that is either the source or the destination of the data may request the data transfer. DMA transfer may be also unsynchronized, that is the transfer will take place continually until the correct number of transfers has occurred. When source or unsynchronized transfers are performed, the DMA channel may begin another transfer immediately after the end of a previous DMA transfer. This allows a complete transfer to take place every 2 bus cycles or 8 clock cycles (assuming no wait states). When destination synchronized transfers are performed, data will not be fetched from the source address until the destination device signals that it is ready to receive it. The DMA controller will relinquish control of the bus after every transfer, the CPU can initiate a bus cycle.

DMA Acknowledge: No explicit DMA acknowledge pulse is provided. Since both source and destination pointers are maintained, a read from a requesting source or a write to a requesting destination should be used as the DMA acknowledge signal. Since the chip-select lines can be programmed to be active for a given block of memory or I/O space, and the DMA pointers can be programmed to point to the same given block, a chip-select line could be used to indicate a DMA acknowledge.

DMA Priority: The DMA channels may be programmed to give one channel priority over the other, or they may be programmed to alternate cycles when both have DMA requests pending. DMA cycles always have priority over internal CPU cycles except between locked memory accesses or word accesses to odd memory locations; also an external bus hold takes priority over an internal DMA cycle. Because an interrupt request cannot suspend a DMA operation and the CPU cannot access memory during a DMA cycle, interrupt latency time will suffer during sequences of continuous DMA cycles. An NMI request, however, will cause all internal DMA activity to halt. This allows the CPU to quickly respond to NMI request.

Timers: The 80186 provides three internal 16-bit programmable timers. Two of them are highly flexible and are connected to four external pins. They can be used to count external events, time external events, generate nonrepetitive waveforms, etc. The third one is not connected to any external pins, and is useful for real-time coding and time delay applications, and also can be used as a prescaler to the other two, or as a DMA request source. The timers are controlled by eleven 16-bit registers in the peripheral control block. Each timer gets serviced every fourth CPU-clock cycle, and thus can operate at speeds up to one-quarter the internal clock frequency (one eighth the crystal rate).

4.1.6 Master and Slave Modes Operations

1. Master Mode Operation

Interrupt Controller External Interface: Five pins are provided for external interrupt sources. One is NMI, non-maskable interrupt which is generally used for unusual events. The other four pins may be configured in any of the following three ways:

1. As four interrupt input lines with internally generated interrupt vectors;
2. As an interrupt line and interrupt acknowledge line pair with externally generated interrupt vectors plus two interrupt input lines with internally generated vectors; and
3. As two pairs of interrupt/interrupt acknowledge lines with externally generated interrupt vectors.

Interrupt Controller Modes of Operation. he basic modes of operation of the interrupt controller are fully nested mode, cascade mode, and special fully nested mode. The controller may be used in a polled mode if interrupts are undesirable. The interrupt controller register model consists of fifteen registers: In-service register; interrupt request register; mask register; priority mask register; interrupt status register; timer, DMA 0,1 control registers; INT0-3 control registers; EOI register, poll and poll status registers.

Master Mode Features: The master mode has the following features: programmable priority, end-of-interrupt command, trigger mode, and interrupt vectoring.

2. Slave Mode Operation

When Slave Mode is used, the internal 80186 interrupt controller will be used as a slave controller to an external master interrupt controller. The internal 80186 resources will be monitored by the internal interrupt controller, while the external controller functions as the system master interrupt controller. Upon reset, the 80186 will be in Master Mode. IN Slave Mode, each peripheral must be assigned a unique priority to ensure proper interrupt controller operation. Therefore, it is the programmer's responsibility to assign correct priorities and initialize interrupt control registers before enabling interrupts. All control and command registers are located inside the internal peripheral control block.

4.2 The Intel 186 Integrated Processor Family Advance Members

The 186 Integrated Processor Family incorporates a wide range of VLSI devices tailored to suit the needs of embedded system designers. All 186 Family devices share a common CPU architecture: the industry standard 8086/8088. Code developed on other "X86" platforms can be ported with little or no modification to any of the 186 Integrated Processor Family devices. Each of the 186 Integrated Processor Family device adds a full complement of peripherals to the 8086/8088 CPU core. The type of peripherals and level of integration vary between family members. A complete 186 Family system can often be designed with just the addition of RAM, ROM and simple glue logic. The space savings afforded by high-integration are critical as designers continue to strive for smaller size and portability.

4.2.1 Intel 80C186 CHMOS High Integration 16-bit Microprocessor

The 80C186 ia a CHMOS high integration microprocessor. It has features which are new to the 80186 family including a DRAM refresh control unit, power-save mode and a direct numerics interface. When used in "compatible" mode, the 80C186 is 100% pin-for-pin compatible with the NMOS 80186 (except for 8087 applications). The "enhanced" mode of operation allows the full feature set of the 80C186 to be used. The 80C186 is upward compatible with 8086 and 8088 software and fully compatible with 80186 and 80188 software.

4.2.2 Intel 80C186XL-20,-16,-12,-10 16-bit High Integration Embedded Processor

The Intel 80C186XL is a Modular Core re-implementation of the 80C186 Microprocessor. It offers higher speed and lower power consumption than the standard 80C186 but maintains 100% clock-for-clock functional compatibility. Packaging and pinout are also identical.

4.2.3 Intel 80C186EA-20,-16,-12 16-bit High Integration Embedded Processor

The Intel 80C186EA is a CHMOS high integration embedded microprocessor. It is the second product in a new generation of low-power, high-integration microprocessors. It enhances the existing 80C186 family by offering new features and new operating modes. The 80C186EA is object code compatible with the 80C186/80C188 embedded processor. It adds the additional capabilities of Idle and Powerdown Modes.In Numerics Mode, the 80C186EA interfaces directly with an 80C187 Numerics Coprocessor.

4.2.4 Intel 80C186EB-20,-16,-13,-8 16-bit High-Integration Embedded Processor

The Intel 80C186EB is a second generation CHMOS High-Integration microprocessor. It has features that are new to the 80C186 family and include a STATIC CPU core, an enhanced Chip Select unit, two independent Serial Channels, I/O ports, and the capability of Idle or Powerdown low power modes. It is object code compatible with the 80C186/80C188 microprocessors.

4.2.5 Intel 80C186EC-16,-13 16-bit High-Integration Embedded Processor

The Intel 80C186EC is one of the highest integration members of the 186 Integrated Processor Family. It uses the latest high density CHMOS technology to integrate several of the most common system peripherals with an enhanced 8086 CPU core to create a compact, yet powerful system on a single monolithic silicon die.

4.2.6 Intel 80L186EA-13, -8 16-bit High Integration Embedded Processor

The 80L186EA is the second member of the 186 Integrated Processor Family to go to

3V operation, following the 80L186EB. The 80L186EA is the 3V version of the 80C186EA. It is functionally compatible with the industry standard 80C186 embedded processor. Current 80C186 users can easily upgrade their designs to use the 80L186EA and benefit from the reduced power consumption of 3V operation. The feature set of the 80L186EA meets the needs of battery-powered applications which benefit from the static CPU core and peripherals. Minimum current consumption is achieved by combining low voltage operation along with features of the power management unit, thus maximizing battery life.

4.2.7 Intel 80L186EB-13, -8 16-bit High-Integration Embedded Processor

The 80L186EB is the 3V version of the 80C186EB embedded processor. It is the first product in a new generation of low-power, high-integration microprocessors. It enhances the existing 186 family by offering new features and new operating modes. It is object code compatible with the 80C186/80C188 microprocessors. By reducing Vcc, further power savings can be realized over the standard 80C186EB, making the 80L186EB ideal for battery-powered portable applications.

4.3 Intel 80C187 80-bit Math Coprocessor

The Intel 80C187 is a high-performance math coprocessor that extends the architecture of the 80C186 with floating-point, extended integer, and BCD data types. It also executes numerous built-in transcendental functions. A computing system that includes the 80C187 fully conforms to the IEEE Floating-Point Standard. The 80C187 adds over 70 mnemonics to the instruction set of the 80C186, including support for arithmetic, exponential, and trigonometric mathematical operations. It is implemented with 1.5 micron, high-speed CHMOS III technology and packaged in both a 40-pin CERDIP and a 44-pin PLCC package, and is also upward object-code compatible from the 8087 math coprocessor and will execute code written for the 80387DX and 80387SX math coprocessor. A 80C186 system that includes the 80C187 is completely upward compatible with software for the 8086/8087. The 80C187 interfaces only with the 80C186 CPU. The interface hardware for the 80C187 is not implemented on the 80C188.

Programming Interface: The 80C187 adds to the CPU additional data types, registers, instructions, and interrupts specifically designed to facilitate high-speed numerics processing. All new instructions and data types are directly supported by the assembler and compilers for high-level languages. All communication between the CPU and the 80C187 is transparent to applications software. The CPU automatically controls the 80C187 whenever a numerics instruction is executed. All physical memory and virtual memory of the CPU are available for storage of the instructions and operands of programs that use the 80C187. All memory addressing modes are available for addressing numerics operands.

Data Types: The 80C187 supports the following 7 data types: word integer, short integer, long integer, packed BCD, single precision real, double precision real, and extended precision real.

Register Set: When an 80C187 is present in a system, programmers may use 80C187's registers in addition to the registers normally available on the CPU. The 80C187's register set includes: 8 data registers R0--R7, control register, status register, tag register, instruction pointer, and data pointer.

Interrupt Description: CPU interrupt 16 is used to report exceptional conditions while executing numeric programs. It indicates that the previous numerics instruction caused an unmasked exception.

Exception Handling: The 80C187 detects six different exception conditions that can occur during instruction execution. They are: Invalid Operation, Denormalized Operand, Zero Divisor, Overflow, Underflow, and Inexact Result (precision).

Signal Description: The 80C187 pins are grouped by function as follows:

1. Execution Control--CLK (Clock), CKM (Clock Mode), RESET (System Reset);
2. NPX Handshake--PEREQ (Processor Extension Request), BUSY (Busy Status), ERROR (Error Status);
3. Bus Interface Pins--D15-D0 (Data Pins), NPWR (Numeric Processor Write), NPRD (Numeric Processor Read);
4. Chip/Port Select--NPS1 (Numeric Processor Select), NPS2 (Numeric Processor Select), CMD0 (Command Select), CMD1 (Command Select);
5. Power Supplies--Vcc (System Power), Vss (System Ground).

Bus Cycles: The pins NPS1, NPS2, CMD0, CMD1, NPRD, and NPWR identify bus cycles for the NPX.

CPU/NPX Synchronization: The pins BUSY, PEREQ, and ERROR are used for various aspects of synchronization between the CPU and the NPX.

4.4 Intel 80188 High Integration 8-bit Microprocessor

The Intel 80188 is a very high integration 8-bit microprocessor. It combines 15-20 of the most common microprocessor system components onto one chip while providing twice the performance of the standard 8088. It is object code compatible with the 8086, 8088 microprocessors and adds 10 new instruction types to the 8086, 8088 instruction set. The 8086, 8088, 80186, 80188 and 80286 family all contain the same basic set of registers, instructions, and addressing modes. The 80188 processor is upward compatible with the 8086, 8088, 80186, and 80286 CPUs.

Register Set: The 80188 has 14 registers: 8 general purpose registers: AX, BX, CX, DX, BP, SI, DI, and SP; 4 segment registers: CS, DS, SS, and ES; and two status and control registers: F, and IP.

Instruction Set: The instruction set is divided into 7 categories: Data Transfer, Arithmetic, Shift/Rotate/Logical, String Manipulation, Control Transfer, High-level Instructions, and Processor Control.

Memory Organization: The memory of 80188 is organized in sets of segments. Each segment is a linear contiguous sequence of up to 64K 8-bit bytes.

Addressing Modes: The 80188 has 8 addressing modes: Register Operand Mode, Immediate Operand Mode, Direct Mode, Register Indirect Mode, Based Mode, Indexed Mode, Based Indexed Mode, and Based Indexed Mode with Displacement.

Data Types: The 80188 directly supports 8 data types: Integer, Ordinal, Pointer, String, ASCII, BCD, Packed BCD, Floating Point.

I/O Space: The 80188 I/O space has 64K 8-bit or 32K 16-bit ports.

Interrupt: The 80188 has the following interrupts: Divide Error Exception, Single-Step Interrupt, Breakpoint Interrupt, Into Detected Overflow Exception, Array Bounds Exception, Unused Opcode Exception, Escape Opcode Exception, and Non-Maskable Interrupt Request.